PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR-MATRIZ
FACULTAD DE CIENCIAS ADMINISTRATIVAS Y CONTABLES

TRABAJO DE INVESTIGACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGÍSTER EN ADMINISTRACIÓN DE EMPRESAS CON MENCIÓN EN GERENCIA DE LA CALIDAD Y PRODUCTIVIDAD

ESTÁNDAR DE PROGRAMACIÓN Y COMPENDIO DE BUENAS PRÁCTICAS EN ORACLE PL/SQL

LIC. LEONARDO DANIEL VENUTI

DIRECTOR: MBA. CECILIA PATRICIA LEÓN VEGA

LÍNEA DE INVESTIGACIÓN: SISTEMAS DE GESTIÓN DE PRODUCCIÓN Y OPERACIONES

QUITO, OCTUBRE 2016

DIRECTORA:

Dra. Cecilia Patricia León Vega, MBA, MSC.

INFORMANTES:

Mgtr. Edwin Suquillo Guijarro

Mgtr. César Augusto Morales Mejía

Agradecimientos

Quiero agradecer a mi amada esposa Adriana Oñate por apoyarme a lo largo de este camino, a Rosario Coronel y Edward Evans por su constante ayuda y a todos mis compañeros de la maestría por ayudarme a finalizar esta etapa luego del accidente sufrido durante la cursada de la misma.

Índice

Índice de figuras

RESÚMEN EJECUTIVO

La finalidad del trabajo es aportar un estándar, una serie de pautas y buenas prácticas para mejorar la calidad de los desarrollos que se realizan en las Bases de Datos Oracle para que el usuario final obtenga soluciones que realicen las consultas a la Base de Datos con rapidez y eficiencia.

La aplicación del estándar de programación permite unificar la manera en que se programa PL/SQL para que al leer el código escrito por otros programadores, su lectura resulte simple y llevadera. La aplicación del compendio de buenas prácticas permite mejorar la calidad del código desarrollado y optimiza el consumo de los recursos de la Base de Datos.

Se inició el trabajo escribiendo tanto el estándar como el compendio de buenas prácticas en base a la experiencia de más de 10 años de haber programado Oracle PL/SQL y luego para validar la aplicabilidad se realizaron encuestas a programadores.

En el primer capítulo, se indican los fundamentos que llevaron a la elaboración tanto del estándar como del compendio de buenas prácticas y que permitirán entender el porqué de este trabajo.

En el segundo capítulo, se describe la terminología que estará presente en todo el documento y permitirá entender los conceptos descriptos.

En el tercer capítulo, se indica la situación actual y se citan estudios sobre el impacto de las bases de datos, tiempos de carga y tasas de abandono tanto en computadoras como en smartphones.

En el cuarto capítulo, se describen las buenas prácticas que a los largo de los años me han ayudado a mejorar los tiempos de ejecución en las consultas realizadas en bases de datos Oracle y así obtener mejores resultados.

En el quinto capítulo, se indican las convenciones que deberán adoptarse a modo de estándar y que permitirán lograr código fuente más legible y fácil de entender y por ende de modificar.

En el sexto y último capítulo, se muestran los resultados obtenidos luego de realizar encuestas a colegas con el fin de validar el trabajo realizado. Se mencionan las conclusiones a las que se llegaron y se dan una serie de recomendaciones para su aplicación.

Los resultados obtenidos son alentadores ya que tanto el Estándar de Programación como el Compendio de Buenas Prácticas son considerados legibles y aplicables por los colegas encuestados, no hubo ninguna persona encuestada que manifestara lo contrario.

INTRODUCCIÓN

Este trabajo tiene por objetivo estandarizar la manera de programar PL/SQL y proporcionar una serie de recomendaciones a modo de buenas prácticas de programación para lograr desarrollos en bases de datos Oracle de mayor calidad y de más fácil legibilidad logrando así código más comprensible y de fácil seguimiento, y por consiguiente aprovechar de mejor manera los recursos del sistema logrando que las ejecuciones de las consultas sean más efectivas.

Como relevancia social, el trabajo aporta un estándar y una serie de pautas y buenas prácticas para mejorar la Calidad de los desarrollos que se realizan en las bases de datos Oracle para que el usuario final obtenga soluciones que realicen las consultas a la base de datos con rapidez y eficiencia.

Como relevancia académica, el trabajo pretende convertirse en un referente a nivel empresa para que los programadores tengan las herramientas necesarias para realizar desarrollos de base de datos Oracle con la mayor efectividad y aprovechando de mejor manera los recursos informáticos disponibles.

Y por último como relevancia personal, con el conocimiento adquirido luego de más de 10 años de programar PL/SQL se pretende ayudar a los programadores a realizar su trabajo de manera más efectiva para que puedan aprovechar todos los recursos que presenta el

Sistema de Gestión de Base de Datos Oracle y así lograr consultas de baja complejidad que resuelvan los requerimientos del usuario utilizando el menor tiempo posible y consumiendo la menor cantidad de recursos disponibles en las bases de datos.

1. FUNDAMENTOS

PL/SQL es un lenguaje muy flexible, ya que se pueden usar distintas variantes de una sentencia para alcanzar el mismo resultado y por lo tanto los programadores suelen creer que mientras la sentencia PL/SQL devuelva el resultado esperado, esta puede considerarse correcta. Pero en realidad, una sentencia PL/SQL es correcta sólo si produce el resultado esperado en el menor tiempo posible de acuerdo con los recursos disponibles en el sistema.

Los programadores deben tener la habilidad y responsabilidad de optimizar cada sentencia PL/SQL para que la aplicación trabaje eficientemente utilizando la menor cantidad de recursos disponibles. Para que esto sea posible se deben seguir ciertas pautas de programación y utilizar buenas prácticas que ayudan a mejorar la calidad del código desarrollado.

Cuando se codifica una aplicación software, existe un número potencialmente infinito de programas que satisfacen los mismos requisitos. Sin embargo, no todos estos programas comparten los mismos atributos de calidad. Es decir, no todos son iguales de eficientes tanto en consumo de procesador como de memoria; no todos son igual de legibles, no todos son igual de fáciles de modificar; no todos son igual de fáciles de probar y verificar que funcionan correctamente.

La labor de un Ingeniero Informático o Ingeniero Software no es solamente crear aplicaciones de software que funcionen correctamente, sino que además debe crear aplicaciones de software robustas, eficientes, fáciles de mantener y un largo etcétera de propiedades extras. Muchas personas están capacitadas para codificar aplicaciones pero un número bastante más reducido de ellas están capacitadas para hacerlo conforme a los parámetros de calidad que se espera de un titulado superior en Informática. Hoy en día es muy fácil aprender a programar por cuenta propia, dado la amplia cantidad de información disponible a través de internet y la multitud de cursos que se ofertan.

Para lograr software de calidad se cree necesario dar una serie de pautas y proporcionar buenas prácticas de programación para lograr desarrollos en Bases de Datos Oracle de mayor calidad, de más fácil legibilidad logrando así código más comprensible y de fácil seguimiento, y aprovechar de mejor manera los recursos del sistema logrando que las ejecuciones de las consultas sean más efectivas.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software y puede darle mantenimiento. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, y/o mejorar el rendimiento. Las buenas prácticas diferencian a los buenos programadores de los malos programadores.

Este trabajo aporta un estándar el cual se describirá en el capítulo 4, una serie de pautas y buenas prácticas que se explicarán en el capítulo 5 y que en conjunto permitirán mejorar la

calidad de los desarrollos que se realizan en las Bases de Datos Oracle para que el usuario final obtenga soluciones que realicen las consultas a la Base de Datos con rapidez y eficiencia. El estándar propuesto pretende convertirse en un referente a nivel empresa para que los programadores tengan las herramientas necesarias para realizar desarrollos de Base de Datos Oracle con la mayor efectividad y aprovechando de mejor manera los recursos informáticos disponibles. Con esto se pretende ayudar a los programadores a realizar su trabajo de manera más efectiva y aprovechando todos los recursos que presenta el Sistema de Gestión de Base de Datos para lograr consultas de baja complejidad que resuelvan los requerimientos del usuario utilizando el menor tiempo posible y consumiendo la menor cantidad de recursos disponibles en las Bases de Datos Oracle.

El principal objetivo de este trabajo es poder unificar la manera en que se programa PL/SQL para que al leer el código escrito por otros programadores, su lectura resulte simple y llevadera. Evitando así retrabajo y por ende pérdida de tiempo. Aplicando este estándar se pretende lograr código compacto y de fácil lectura, de esta manera se eleva el nivel de productividad del programador, se minimiza el retrabajo y se evita la pérdida de tiempo tratando de entender lo realizado por otro programador. También se logra un producto final de mayor calidad y con una menor taza de errores.

Aplicando en conjunto tanto el estándar de programación como el compendio de buenas prácticas se pretende lograr lo siguiente:

- Establecer reglas y convenios
- Estandarizar el desarrollo

- Aportar higiene y legibilidad al código
- Lograr código de fácil lectura, fácil comprensión y fácil seguimiento, fácil lectura es igual a fácil entendimiento. Al mejorar la legibilidad del código, se permite a otros programadores entender código desarrollado por otros con mayor rapidez y profundidad
- Facilitar la escalabilidad del código
- Facilitar la reutilización y la integración de manera homogénea
- Lograr el mayor equilibrio entre código de concurrencia de grano fino, es decir, se aprovecha de la ejecución de operaciones concurrentes a nivel del procesador (hardware) y código de grano grueso, es decir, se aprovecha de la ejecución de procesos o aplicaciones a nivel del sistema operativo o a nivel de la red de computadoras
- Disminuir el acoplamiento del código, es decir, un acoplamiento fuerte significa que los Stored Procedures relacionados necesitan saber detalles internos unos de otros, los cambios se propagan por el sistema y el sistema es más difícil de entender y por ende de mantener
- Aumentar la cohesión del código, es decir, examinar los Stored Procedures y decidir si todo su contenido está directamente relacionado con el nombre del Stored Procedure y descrito por el mismo y determinar la medida que indica si un Stored Procedure tiene una función bien definida dentro del sistema
- Un componente de código debe ejecutar una única y bien definida tarea, eso se llama principio de responsabilidad única
- Disminuir el retrabajo y por ende los tiempos de entrega final de los desarrollos
- Disminuir los tiempos que utilizan las consultas para retornar los datos

- Reducir la cantidad de errores que se presentan al corregir código realizado por otros programadores
- Promover la uniformidad en el criterio utilizado por los programadores que conforman un equipo
- Obtener productos de mayor calidad y que consuman menos recursos del sistema
- Facilita a un programador la modificación de tu propio código fuente aunque no se esté trabajando en equipo
- Definir la forma en que deben ser declaradas las variables, las clases, los comentarios
- Especificar qué datos deben incluirse acerca del programador y de los cambios realizados al código fuente

2. MARCO TEÓRICO Y GLOSARIO

2.1. Marco Teórico

A continuación se presenta el marco teórico que integra el tema esta investigación con las teorías y enfoques teóricos que se refieren al problema de la presente investigación

Según la Organización Internacional para la Estandarización (ISO), la normalización u estandarización es la actividad que tiene por objeto establecer, ante problemas reales o potenciales, disposiciones destinadas a usos comunes y repetidos, con el fin de obtener un nivel de ordenamiento óptimo en un contexto dado, que puede ser tecnológico, político o económico.

El Instituto Proinapsa nos dice que en general el concepto de buenas prácticas se refiere a toda experiencia que se guía por principios, objetivos y procedimientos apropiados o pautas aconsejables que se adecuan a una determinada perspectiva normativa o a un parámetro consensuado, así como también toda experiencia que ha arrojado resultados positivos, demostrando su eficacia y utilidad en un contexto concreto.

Javier Morales Carreras en su libro Optimización SQL en Oracle, una guía práctica, detallada y completa sobre cómo implementar y explotar bases de datos Oracle de forma eficiente del año 2013, recoge todos los aspectos relativos al rendimiento, útiles tanto para desarrolladores como para administradores de base de datos, y supone una guía básica para entender qué ocurre en el interior de la base de datos y cómo conseguir que la ejecución de

SQL sea óptima. Indica que “el módulo de Oracle responsable de analizar la mejor forma de optimizar una sentencia es el optimizador. El objetivo final del optimizador es la generación de un plan de ejecución óptimo. En sí, el plan de ejecución consiste en la definición de un conjunto de operaciones de acceso y procesamiento de los distintos objetos implicados. Cada vez que una sentencia SQL se procesa en una base de datos, el primer paso es optimizar la consulta y buscar entre distintos planes de ejecución posibles cuál sería la forma más eficiente de ejecutarla. A no ser que el motivo por el cual empecemos a optimizar sentencias sea por quejas sobre una consulta en concreto, lo más común será que ciertas partes de una aplicación empiecen a ralentizarse a medida que va aumentando el volumen de datos que contienen las tablas y los usuarios se quejen a los responsables de la base de datos y de la aplicación.”

Steven Feuerstein en su libro Oracle PL/SQL Best Practices del año 2008, presenta ideas que hacen la diferencia entre un proyecto exitoso y uno que no. Va más allá de la simple enumeración de un conjunto de reglas, y proporciona escenarios realistas que ayudan al lector a entender de dónde vienen las reglas. Ofrece respuestas prácticas a algunas de las preguntas más difíciles que enfrentan los desarrolladores de PL/SQL, incluyendo ¿Cuál es la mejor manera de escribir la lógica SQL en mi código de aplicación? ¿Cómo debo escribir mis paquetes para que puedan ser apalancados por todo mi equipo de desarrolladores? ¿Cómo puedo asegurarme de que todos los programas de mi equipo manejen y registren errores de forma consistente? Incluye las mejores prácticas de Oracle PL / SQL en categorías principales como: desarrollo general de PL/SQL, estándares de programación; variables y estructuras de datos, lógica de control, manejo de errores, uso de SQL en PL/SQL, procedimientos de compilación, funciones, paquetes y disparadores y el rendimiento general del programa.

2.2. Glosario

A continuación se describe la terminología que estará presente en todo el documento y permitirá entender de mejor manera los conceptos descriptos en este trabajo.

Estándar de programación: es una forma de normalizar la programación de forma tal que al trabajar en un proyecto, cualquier persona involucrada en el mismo tenga acceso y comprenda el código.

Buenas prácticas de programación: acciones o iniciativas con repercusiones tangibles y mesurables que han tenido buenos resultados en un determinado contexto, con lo cual se espera que en contextos similares tengan resultados parecidos y que pueden servir como modelos para que otros programadores puedan conocerlos y aplicarlos.

Tiempo de respuesta: el tiempo que transcurre desde que se envía una petición a la Base de Datos y se recibe la respuesta completa.

Porcentaje de rebote: es el porcentaje de visitantes de un sitio web en particular que navegan fuera del sitio después de ver sólo una página.

Base de datos relacional: es una base de datos digital cuya organización se basa en el modelo relacional de datos que consta de una colección de elementos de datos organizados

como un conjunto de tablas formalmente descritas a partir de las cuales se puede acceder a los datos.

Sistema de Gestión de Base de Datos: colección de datos relacionados entre sí, estructurados y organizados, y un conjunto de programas que acceden y gestionan esos datos. La colección de esos datos se denomina Base de Datos.

RDBMS: Relational DataBase Management System o Sistema de Gestión de Base de Datos Relacional o mayormente conocido como motor de la Base de Datos

Oracle: es una de las mayores compañías de software del mundo, sus productos van desde bases de datos hasta sistemas de gestión. Es la empresa creadora de la primer base de datos relacional.

SQL: (Structured Query Language o Lenguaje Estructurado de Consulta) es un lenguaje de programación estandarizado que se utiliza para la obtención de información desde una base de datos, para actualizarla y realizar diversas operaciones sobre los datos en ellas.

PL/SQL: (Procedural Language/Structured Query Language o Lenguaje Procesal/Lenguaje Estructurado de Consulta) es la extensión de procedimientos de Oracle para SQL y la base de datos relacional de Oracle. Incluye elementos de lenguaje procedural tales como condiciones y bucles. Permite la declaración de constantes y variables, procedures (procedimientos) y functions (funciones), types (tipos) y variables de esos tipos, y triggers

(disparadores).

Bind Variable: es una variable de marcador de posición en una sentencia SQL que debe ser reemplazada con un valor válido (o la dirección de un valor) antes que la instrucción pueda ejecutarse correctamente.

SGA: System Global Area o Área Global del Sistema, es la zona de memoria en la que la Base de Datos Oracle guarda información sobre su estado. Esta estructura de memoria está disponible para todos los procesos y es por eso se dice que está compartida.

Query: es el principal mecanismo para recuperar información de una base de datos y consiste en preguntas presentadas a la base de datos en un formato predefinido dentro de un grupo de sentencias SQL que se ejecutan con el fin de obtener datos.

Log: es un historial de acciones ejecutadas por un sistema de gestión de bases de datos utilizado para garantizar la consistencia e integridad.

Variable: es un valor que puede cambiar, dependiendo de las condiciones o de la información transmitida al programa.

Parámetro: es un elemento de información, como un nombre, un número o una opción seleccionada que se pasa a un programa y afectan el funcionamiento del programa que los recibe.

Stored Procedure: el programa se guarda o almacena en el motor de Base Datos, esto permite que cuando el procedimiento sea invocado este se ejecute directamente en el Servidor de la Base Datos de Oracle logrando así una respuesta mucho más rápida.

Package: en Oracle PL/SQL un package o paquete es un grupo de construcciones programáticas combinadas "empaquetadas" en una sola unidad. Consta de dos partes, Package Specification que contiene las construcciones públicas que actúa como la interfaz del paquete y solamente contiene las construcciones de prototipos sin ningún código lógico, y Package Body que contiene la definición de las construcciones de los prototipos de la especificación y también puede contener las unidades de programa privadas o definidas localmente que pueden utilizarse solamente dentro del ámbito del cuerpo de dicho paquete.

Procedure: es un subprograma PL/SQL nombrado que puede opcionalmente aceptar parámetros de entrada y puede o no devolver un valor. Su principal función es integrar un proceso de lógica de negocio y realizar la manipulación de datos con la ayuda de los datos suministrados. Puede devolver valores al entorno solamente a través de los parámetros de salida.

Function: es un conjunto de sentencias PL/SQL que se llaman por nombre. Las funciones almacenadas son muy similares a los procedimientos, excepto que una función devuelve un valor al entorno en el que se llama.

Parse: proceso en que se analiza una secuencia de símbolos a fin de determinar su estructura gramatical con respecto a una gramática formal dada, formalmente es llamado análisis de sintaxis. Se analiza una oración o declaración de lenguaje donde se analizan las palabras en unidades funcionales que pueden convertirse en lenguaje de máquina.

Fetch: retornar registros de datos en la ejecución de un query. Es un comando utilizado en el lenguaje de consulta estructurado (SQL) para recuperar las filas secuencialmente. En SQL, un cursor es un puntero a una fila seleccionada en una colección recuperada por una instrucción SQL. El cursor avanza por las filas, una a la vez, para permitir el procesamiento secuencial de registros. El comando fetch recupera la fila seleccionada del cursor.

Sort: sirve para ordenar los registros retornados en la respuesta a una consulta de una manera particular. Utiliza un parámetro que indica a la base de datos cómo ordenar las filas devueltas. Los parámetros válidos son ASC (ascendente) o DESC (descendente) que especifican el orden ascendente y el orden descendente

Merge: seleccionar las filas de una o más fuentes de actualización o inserción en una tabla o vista. Se seleccionan filas de una o más fuentes para actualizar o insertar en una tabla o vista. Se pueden especificar condiciones para determinar si desea actualizar o insertar en la tabla o vista de destino. Esta declaración es una forma conveniente de combinar múltiples operaciones.

Join: combinación de columnas de una o más tablas en una base de datos relacional en el que se crea un conjunto que puede ser guardado como una tabla o utilizado tal como es y que se utiliza como medio para combinar columnas de una (auto tabla) o más tablas mediante el uso de valores comunes en cada una. Se utiliza para combinar registros de dos o más tablas en una base de datos.

Full table scan: exploración de tabla completa también conocida como exploración secuencial es una exploración hecha en una Base de Datos donde cada fila de la tabla se lee en un orden secuencial. Es una exploración realizada en una base de datos donde cada fila de la tabla en escaneo se lee en un orden secuencial y las columnas encontradas se comprueban para la validez de una condición. Suelen ser el método más lento de escanear una tabla debido a la gran cantidad de lecturas de entrada y salida requeridas desde el disco, que consiste tanto en varias búsquedas como en costosas transferencias de disco a memoria.

Index scan: exploración de tabla mediante índices. El motor de búsqueda puede encontrar la primera fila que coincidirá haciendo una búsqueda basada en el árbol del índice, y luego entonces puede navegar el índice en orden hasta que llegue al final del rango

Hash join: unión de tablas mediante un algoritmo. La base de datos utiliza una combinación hash para unir conjuntos de datos más grandes. El optimizador utiliza el menor de dos conjuntos de datos para crear una tabla hash en la clave de combinación en memoria, utilizando una función de hash determinista para especificar la ubicación en la tabla hash en la que almacenar cada fila. La base de datos entonces explora el conjunto de datos más grande, probando la tabla hash para encontrar las filas que cumplen la condición de combinación.

Driving table: tabla pivote o principal que se utiliza para realizar un join con otras tablas. La tabla pivote es la primera tabla unida, la que devolverá el menor número de filas, y por lo tanto, la que menos registros pasará a las uniones posteriores.

DBA: Data Base Administrator o Administrador de Base de Datos, dirige o realiza todas las actividades relacionadas con el mantenimiento y funcionamiento de un entorno de base de datos exitoso.

DML: Data Manipulation Languaje o Lenguaje de Manipulación de Datos. Las sentencias DML son aquellas sentencias SQL que permiten manipular los datos de una base de datos. Las instrucciones SQL que están en la clase DML son INSERT, UPDATE y DELETE.

DDL: Data Definition Languaje o Lenguaje de Definición de Datos. Se utilizan para definir la estructura de la base de datos. Cualquier comando CREATE, DROP y ALTER son ejemplos de sentencias DDL SQL.

CamelCase (Camellado): es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello.

DataWarrehouse: es una base de datos corporativa que se caracteriza por integrar y depurar información de una o más fuentes distintas, para luego procesarla permitiendo su análisis desde infinidad de perspectivas y con grandes velocidades de respuesta. La creación de un

datawarehouse representa en la mayoría de las ocasiones el primer paso, desde el punto de vista técnico, para implantar una solución completa y fiable de Business Intelligence.

Paginar: es uno de los esquemas de manejo de memoria en donde una computadora puede almacenar y recuperar datos de un dispositivo de almacenamiento secundario para su uso en la memoria principal.

Tabla: es una colección de datos relacionados mantenidos en un formato estructurado dentro de una base de datos. Consiste de columnas y de filas. En una base de datos relacional es un conjunto de elementos de datos (valores) que utilizan un modelo de columnas verticales (identificables por nombre) y filas horizontales, siendo la celda la unidad donde una fila y una columna se cruzan.

Vista: es una forma de retratar la información en la base de datos. Es un objeto de búsqueda en una base de datos que está definida por una consulta. Puede combinar datos de dos o más tablas utilizando joins, y también contiene un subconjunto de información.

Índice: es una estructura de datos que mejora la velocidad de las operaciones de recuperación de datos en una tabla de base de datos a costa de escrituras adicionales y espacio de almacenamiento para mantener la estructura de datos de índice. Los índices pueden ser simples o compuestos, son simples cuando están conformados por un solo campo y compuestos cuando están conformados por más de un campo.

Trigger: es un procedimiento que se ejecuta automáticamente en respuesta a determinados eventos en una tabla o vista en particular en una base de datos. El trigger o disparador se utiliza principalmente para mantener la integridad de la información en la base de datos. Permite tomar acciones ante los eventos que suceden en tablas o vistas.

Esquema: se refiere a la organización de los datos como un modelo de cómo se construye la base de datos. Es una colección de estructuras lógicas de datos u objetos de esquema. Un esquema es propiedad de un usuario de base de datos y tiene el mismo nombre que ese usuario. Cada usuario posee un único esquema. Los objetos de esquema se pueden crear y manipular con sentencias SQL.

Diccionario de Datos: es quizá una de las partes más importantes de Oracle. Se trata de un conjunto de tablas de sistema, de sólo lectura, que proporcionan información muy útil sobre la base de datos. Contiene las definiciones de todos los objetos de esquema de la base de datos (tablas, vistas, índices, procedimientos, funciones, paquetes, etc.) y también cuánto espacio ha sido asignado y utilizado actualmente por los objetos de cada esquema.

Execution Plan (Plan de Ejecución): muestra los planes de ejecución elegidos por el optimizador de Oracle para las sentencias SELECT, UPDATE, INSERT y DELETE. El plan de ejecución de una sentencia es la secuencia de operaciones que Oracle realiza para ejecutar una sentencia. Los componentes de los planes de ejecución incluyen una ordenación de las tablas referenciadas por la sentencia, un método de acceso para cada tabla mencionada en la declaración y un método de combinación para tablas afectadas por operaciones de combinación en la sentencia.

SQL Dinámico: es una metodología de programación para generar y ejecutar sentencias SQL en tiempo de ejecución. Es útil para escribir programas de propósito general y flexibles como sistemas de consulta, al escribir programas que deben ejecutar instrucciones de lenguaje de definición de base de datos (DDL) o cuando no se sabe en tiempo de compilación el texto completo de una sentencia SQL o el número o tipos de datos de sus variables de entrada y salida.

Foreign Keys (Clave Foránea): es un campo o colección de campos en una tabla que identifica de forma única una fila de otra tabla o la misma tabla. En palabras más simples, la clave foránea se define en una segunda tabla, pero se refiere a la clave principal en la primera tabla. La tabla que contiene la clave foránea se denomina tabla secundaria y la tabla que contiene la clave candidata se denomina tabla referenciada o primaria.

Cursor: es una sentencia SELECT que se define dentro de la sección de declaración de código PL/SQL. Para ejecutar una consulta de varias filas, Oracle abre un área de trabajo sin nombre que almacena la información de procesamiento. Un cursor permite nombrar el área de trabajo, acceder a la información y procesar y recorrer las filas individualmente para tomar acciones sobre ellas.

Vector: es una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico se puede ver como un conjunto de elementos ordenados en fila o filas y columnas si tuviera dos dimensiones.

Excepción: es todo error PL/SQL que sucede durante la ejecución de un programa, implícitamente por la base de datos o explícitamente por un programa. Existen tres tipos de excepciones, las excepciones predefinidas que son las condiciones de error definidas por PL/SQL, las excepciones no predefinidas que incluyen cualquier error en la Base de Datos y las excepciones definidas por el usuario que son excepciones específicas de la aplicación realizada por un programador.

Compilar: es el proceso de crear un programa ejecutable de código escrito en un lenguaje de programación compilado. La compilación permite a la computadora ejecutar y comprender el programa sin necesidad del software de programación utilizado para crearlo. Cuando un programa se compila a menudo se compila para una plataforma específica.

3. SITUACIÓN ACTUAL

En la era digital que vivimos los tiempos de respuesta de las aplicaciones que realizan consultas pesadas a las Bases de Datos deben tender a reducirse. Se entiende por velocidad de carga de una aplicación, el tiempo de espera transcurrido desde que se envía la primera petición hasta que se recibe la última respuesta, dicho de otra manera, es el tiempo transcurrido desde que se utiliza una opción de una aplicación hasta que esta deja de cargar por completo. Los tiempos de respuesta pueden variar sensiblemente si las consultas a las Bases de Datos están mal realizadas y por ende no responden en un tiempo prudencial.

El tiempo de carga de las páginas web es obviamente una parte importante de la experiencia del usuario de cualquier sitio web. Por desgracia, los visitantes del sitio tienden a preocuparse más por la velocidad que por todas las funcionalidades y ventajas que tenga un sitio web. Además, el tiempo de carga de la página se está convirtiendo en un factor más importante cuando se trata de posicionamiento en los buscadores como Google, Yahoo o Bing.

A lo largo de este capítulo se describen tres factores distintos que inciden en los tiempos de respuesta y se citan estudios que demuestran el impacto o incidencia de estos factores. Factores que se pretende reducir o mitigar con la aplicación del estándar propuesto en el capítulo 4 y de las buenas prácticas enumeradas en el capítulo 5.

3.1. Impacto de las Bases de Datos

A lo largo de lo años he podido observar que los problemas en las aplicaciones a menudo se producen debido al rendimiento de la base de datos. Un rendimiento pobre de la base de datos se debe al alto tiempo de espera de la consulta, que a su vez hace que otras consultas se ejecuten lentamente. Mediante el seguimiento del tiempo de respuesta de las consultas en la base de datos y la identificación de cuellos de botella, se puede asegurar que el rendimiento de la base de datos sea óptimo y no afecte al rendimiento de las aplicaciones que dependen de ella.

La mayoría de los sitios web utilizan bases de datos para almacenar información. Por lo general tiendas de e-commerce, blogs, sitios de noticias, o cualquier tipo de funcionalidad dinámica con búsquedas internas utilizan una base de datos. El rentimiento de la base de datos puede afectar la velocidad de la página. Una gran parte de la razón por la que los sitios se vuelven lento es porque necesitan ser optimizados. La mayoría de los sitios web de hoy son sitios dinámicos, esto significa que el contenido del sitio web se genera al menos parcialmente cuando un visitante solicita la página. Muchas veces, el contenido generado depende de una base de datos. Si la base de datos o el código que interactúa con la base de datos y que genera la página para sus visitantes no está optimizada, puede provocar un retraso en los tiempos de carga de la página.

Según revisiones escritas por la comunidad IT Central Station, el principal factor que afecta el tiempo de respuesta del sitio web es la lenta consulta a las bases de datos. Algunos de los revisores recomiendan analizar las consultas que se ejecutan, asegurándose que los índices se están utilizando correctamente. De esta forma, se puede reducir el

tiempo empleado en la base de datos, mejorando así el tiempo de respuesta. (Rothstein, Russell, 2016)

Cuando los usuarios experimentan un tiempo de respuesta deficiente en el sitio web, los especialistas necesitan profundizar más allá de la red o las aplicaciones. Al cargar, es posible que una página web no muestre errores, pero los usuarios podrían estar esperando contenido dinámico clave como un saldo bancario o el estado de pedido. Este problema frecuentemente no es el servidor web o en las aplicaciones, sino en la base de datos que los soporta. El código SQL incorrecto y las bases de datos no optimizadas representan hasta dos tercios de todos los problemas de rendimiento del sitio web. Si se optimizan las consultas SQL mejorará el rendimiento del sitio web para que la gente vuelva a culpar a la red. (Robert Anderson, 2015)

En Stackify indican que uno de los principales factores que han visto en su análisis de cientos de aplicaciones es la interacción con las bases de datos; ya sea consultas de base de datos lentas o demasiadas consultas de base de datos que se ejecutan por solicitud web. A medida que las empresas analizan cómo su código pasa su tiempo, es esencial ver cuántas consultas se llaman y cuánto tiempo tardan en ejecutarse. (Matt Watson, 2015)

3.2. Tiempos de carga en computadoras

Según estudios realizados sobre los tiempos de respuesta de páginas web, el porcentaje de rebote que sufre una página está relacionado con el tiempo de carga que esta conlleva. Si se quiere que un visitante se quede navegando por la página web, lo primero que se debe

hacer es facilitarle las cosas. Según un estudio realizado por Borland, las navidades y en especial las temporadas de compras masivas como Black Friday y Cyber Monday suelen ser una de las mejores temporadas para el comercio, ya sea tradicional u online. Por ello, estudiar el comportamiento de los clientes en estas fechas resulta fundamental para saber los aciertos y errores que se pueden cometer. Los resultados de este estudio demuestran que el tiempo de espera mientras se carga el sitio web donde los usuarios quieren comprar es una de las claves.

El estudio demuestra que si un sitio web tarda mucho en cargar, los usuarios lo abandonan y no realizan la compra y concluye que muchos sitios de comercio electrónico no supieron gestionar el aumento de la demanda lo que provocó largos tiempos de espera en la carga de sus páginas web y por ende que muchas compras no se realizaran. Según este estudio, el comprador medio por internet está dispuesto a esperar como mucho dos segundos a que cargue la página y, después de tres segundos un 40% de los usuarios abandona la página. Si la compra se está realizando desde un dispositivo móvil, el usuario no está dispuesto a esperar más de cinco segundos a que cargue la página y, superado este tiempo, el 74 % decide abandonar ese sitio. Por ejemplo, un segundo de retraso en el tiempo de respuesta de una página equivale a un 11% menos de páginas visitadas y un 16% de disminución en el grado de satisfacción del cliente. Una vez que los usuarios abandonan un sitio Web resulta muy difícil que vuelvan. De hecho, según los datos de este estudio, el 88 % de los internautas está menos dispuesto a regresar a una página después de una mala experiencia**.** "Hay una gran cantidad de datos que muestran como los usuarios pierden la paciencia con los pobres rendimientos de las páginas Web", ha afirmado Archie Roboostoff, director de producto en Borland. En su opinión, "esto muestra como un importante número de páginas web han perdido ingresos potenciales, durante el periodo navideño, como resultado de la

incapacidad de su sitio para procesar altos niveles de tráfico". (Borland Software Corporation, 2013)

En un análisis realizado por mi persona utilizando la herramienta de medición Google Analytics sobre un sitio de servicios en línea de una importante empresa de telecomunicaciones se pudo determinar que durante el 2014 hubo 2766 sesiones promedio por día y que la durante todo el año la tasa de rebote fue del 29% en promedio, es decir unas 802 sesiones finalizaron abruptamente antes que la página terminara de cargar. Con esto podemos inferir que 3 de cada 10 personas cerraron la página antes que esta finalizara su carga completa y para un sitio web que pretende brindar servicios en línea a los clientes eso es mucho. El tiempo de carga de la página web varía entre 12 y 15 segundos, la meta de la empresa es bajar el tiempo de carga a entre 4 y 5 segundos y con esto se espera reducir la tasa de rebote al 5%. Para realizar esto la empresa está realizando las siguientes actividades:

- Reingeniería del Web Site aplicando Buenas Prácticas de Programación
- Reingeniería del Web Service que interactúa con los sistemas principales de la empresa, aplicando Buenas Prácticas de Programación
- Migración de la Base de Datos a un Sistema de Gestión de Base de Datos con mayores prestaciones
- Migración de la aplicación a un servidor con mayores prestaciones
- Aplicación de sistemas redundantes con balanceo de carga

Según encuestas realizadas por Akamai y Gomez.com, casi la mitad de los usuarios de Internet esperan que un sitio cargue en 2 segundos o menos, y tienden a abandonar un sitio que no está cargado en 3 segundos. El 79% de los compradores de Internet dicen que tienen problemas con el rendimiento de los sitios web y que no van a volver al sitio para comprar de nuevo, y alrededor del 44% de ellos les diría a un amigo si tuvieron una mala experiencia de compras en línea. Esto significa que no sólo se están perdiendo visitantes actualmente en un sitio, pero que la pérdida se magnifica a sus amigos y colegas. El resultado final, es que un montón de potencial de ventas se vaya por el desagüe a causa de unos muy pocos segundos de diferencia, lo cual es muy lamentable y se puede prevenir utilizando las técnicas correctas. (Work, Sean)

Otro estudio realizado por Forrester Consulting indica que el pobre rendimiento de un sitio web conduce a la insatisfacción con más frecuencia que cualquier otro factor. El 64% de los banqueros y corredores de bolsa en línea de Estados Unidos han tenido una experiencia insatisfactoria al dar servicio a sus cuentas, el rendimiento de los sitios web es de lejos el mayor motivo de esta insatisfacción. Como las tareas en línea se vuelven más urgentes o complejas, los usuarios son menos propensos a intentar más tarde y por lo tanto hay más probabilidades de pasarse a canales más caros para completar las transacciones. El 56% de los bancos en línea se trasladaría a canales fuera de línea, el 54% de los corredores de bolsa en línea se movería a los canales fuera de línea para realizar intercambio de inversiones si el sitio no está disponible o es muy lento para responder y el 48% de los banqueros y corredores en línea, dijo que el mal desempeño tuvo un impacto significativo en su verosimilitud para recomendar los servicios de una empresa a un amigo o familiar. (Joshua Bixby, 2010)

Otro estudio realizado por PhoCusWright revela que el 57% de los consumidores online abandonará un sitio web después de esperar 3 segundos para que se cargue la página, 8 de cada 10 personas no va a volver a un sitio después de una experiencia decepcionante y de ellos, 3 le dirá a otros acerca de su mala experiencia lo que acrecentará la cantidad total de usuarios descontentos. (Joshua Bixby, 2010)

En cuanto a la afectación de las compras online Aberdeen Group indica que 1 segundo de retraso en el tiempo de carga de página es igual a 11% menos de páginas vistas, una disminución del 16% en la satisfacción del cliente, y la pérdida del 7% en las compras. En término de dólares, esto significa que si su sitio normalmente gana $100.000 al día, durante un año podría perder $2.5 millones en ventas. (Joshua Bixby, 2010)

Strangeloop nos revela que un sitio que se carga en 3 segundos experimenta 22% menos de páginas vistas y un porcentaje de abandonos del 50% más alto que un sitio que se cargue en 1 segundo. El impacto en las ventas es del -22%. Un sitio que se carga en 5 segundos experimenta 35% menos de páginas vistas y una tasa de rebote del 105% y el impacto en las ventas es del -38%. Finalmente un sitio que se carga en 10 segundos experimenta 46% menos de páginas vistas y una tasa de rebote del 135% y el impacto en las ventas es del -42%. (Joshua Bixby, 2010)

Shopzilla bajó su promedio de tiempo de carga de página de 6 segundos a 1,2 segundos y experimentó un aumento del 12% en ingresos y un aumento del 25% en páginas vistas. También se duplicó el número de sesiones de marketing de motores de búsqueda y gracias

a aplicar Buenas Prácticas de Programación redujo el número de servidores utilizados a la mitad. (Joshua Bixby, 2010)

Amazon encontró que aumentó sus ingresos en un 1% por cada 100 milisegundos de mejora. Mozilla redujo 2,2 segundos de carga en sus páginas de destino, lo que aumentó la transferencia de descargas en un 15,4%, que se estiman resultará en 60 millones más descargas de Firefox por año. Yahoo aumentó el tráfico en un 9% por cada 400 milisegundos de mejora. Según AOL los visitantes en el top ten del percentil de la velocidad del sitio vieron 50% más páginas que los visitantes en la parte inferior de decimo percentil. (Joshua Bixby, 2010)

En el siguiente gráfico puede observarse que los tiempos de carga medios de un sitio web en las sesiones con rebotes son aproximadamente 2,4 segundos superiores a los de las sesiones sin rebotes. Mientras más rápido se carga una página menos rebote tendrá la misma.

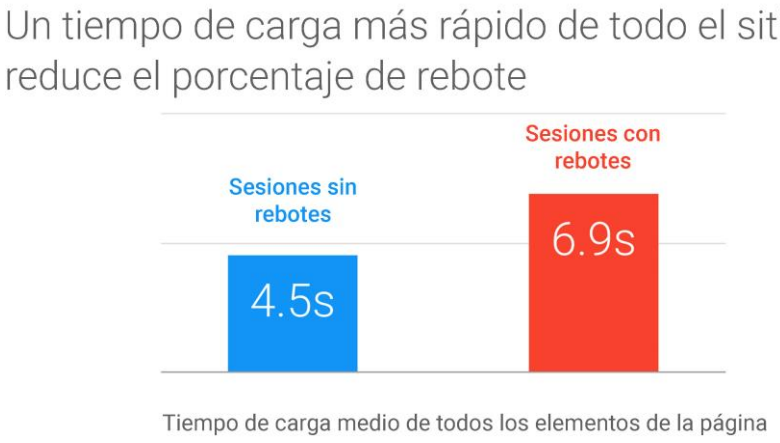

Figura 1. Tiempo de carga más rápido reduce porcentaje de rebote
Fuente: Estudio de Google/SOASTA

Este otro gráfico nos demuestra que cuanto más tiempo un usuario tiene que esperar a que se cargue la página, más probabilidades hay de que abandone esa página.

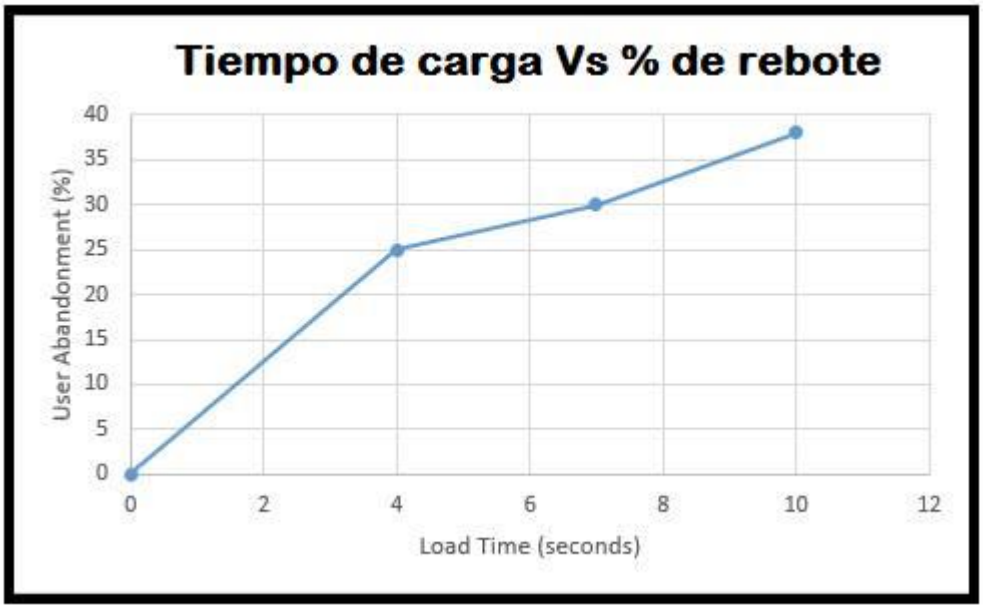

Figura 2. Tiempo de carga VS porcentaje de rebote
Fuente: http://www.socialetic.com

3.3. Tiempos de carga en smartphones

Con las tecnologías actuales, no podemos dejar de lado los smartphones. La experiencia de los usuarios varía según la tecnología con la que estén navegando. La velocidad utilizando tecnología 2G es inferior a 3G la cual a su vez es inferior a 4G, lo que hace que los usuarios no obtengan los mismos tiempos de respuesta. Más y más usuarios están accediendo a los sitios web a través de smartphones y otros dispositivos móviles. Si bien un dispositivo móvil es muy diferente a una computadora, las expectativas de los usuarios siguen siendo las mismas. La tecnología móvil nunca ha sido tan crucial para el éxito del

negocio como lo es ahora. Más de la mitad del tiempo dedicado a los sitios de venta se lleva a cabo en un dispositivo móvil. El comprador en línea promedio realiza 6,2 visitas a la página web de una empresa, usando 2.6 dispositivos, antes de comprar. Sólo un segundo de retraso en los tiempos de carga móviles puede reducir las compras hasta en un 3,5%. Es por esto que los sitios tienen que ejecutarse de forma rápida y consistente en todas las plataformas. Es fundamental para los propietarios del sitio tener visibilidad del rendimiento móvil de sus sitios ya que un servicio móvil que es lento, o peor aún que no funciona, repercute negativamente en los ingresos, aumenta costos y daña el valor de la marca.

De acuerdo con un estudio realizado por Compuware, el 71% de los usuarios de dispositivos móviles espera que los sitios móviles carguen tan rápido como lo harían en sus computadoras. Una mala experiencia en un sitio web móvil hace que los usuarios de dispositivos móviles no vuelvan a visitar o recomendar, un sitio web en particular. Casi la mitad de los usuarios de sitios móviles están poco dispuestos a volver a un sitio web en el que hayan tenido problemas para acceder desde su teléfono, y el 57% es poco probable que recomendar el sitio. (Globe Newsire, 2011)

Según Google, en un teléfono móvil una página web toma 9 segundos para cargar, los tiempos de carga en las computadoras es en promedio unos 6 segundos en todo el mundo, y cerca de 3,5 segundos en promedio Estados Unidos. Los teléfonos móviles tienen mayor latencia y la velocidad de conexión es más lenta que en una computadora. (Bryan McQuade, 2013)

Según una encuesta llevada a cabo por Equation Research en nombre de Compuware:

- El 71% de los usuarios de teléfonos móviles esperan que un sitio web se cargue tan rápido en sus teléfonos como en sus computadoras
- El 74% de los usuarios de teléfonos móviles sólo esperará 5 segundos o menos para que cargue una página
- El 77% de los sitios móviles de las principales compañías tardan más de 5 segundos en cargar
- El 57% de los usuarios de Internet móvil tuvo problemas para acceder a un sitio web
- El 57% de los usuarios de Internet móvil no va a recomendar el sitio web
- El 43% de los usuarios de Internet móvil es poco probable que vuelvan a un sitio que carga lentamente

(Web Performance Guru, 2011)

Según Forrester Consulting 16% de los consumidores ha comprado a través de teléfonos móviles o smartphones, pero 27% de ellos informan que es insatisfactoria debido a la experiencia de compra móvil es demasiado lenta. Un tercio de los consumidores reportan que quieran realizar sus compras a través de sus teléfonos inteligentes en el futuro, con un 5% indica que este será un aspecto importante de su lealtad a las tiendas en línea. (Joshua Bixby, 2010)

Otro estudio de Equation Research nos revela los siguientes datos:

- Más de la mitad de los usuarios de dispositivos móviles dicen que esperan que los sitios sean tan rápido en sus dispositivos móviles como lo hacen en sus computadoras personales.

- 60% de los usuarios de Internet móvil ha tenido un problema al acceder a un sitio web y lento tiempo de carga es el problema número uno que enfrenta casi 3/4 partes de ellos
- Tres de cada 5 dicen que los malos resultados los hará menos probable que vuelva al sitio
- 40% dijo que probablemente la próxima vez visiten el sitio de un competidor
- La mayoría de los participantes en el estudio, dijo que esperan completar transacciones sencillas completas como controlar el balance de su cuenta bancaria en un minuto o menos, o que abandonará el sitio.

(Joshua Bixby, 2010)

En el siguiente gráfico puede verse que los porcentajes de rebote son más altos en los smartphones que en las computadoras.

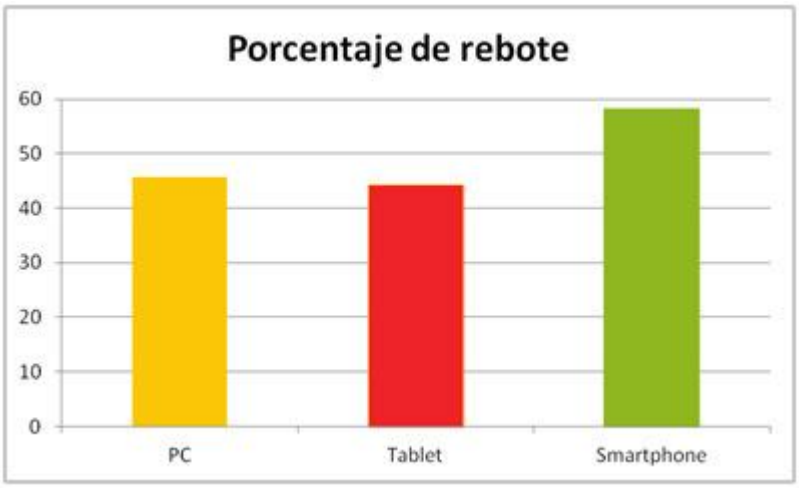

Figura 3. Porcentaje de rebote según dispositivo
Fuente: http://www.lamagnetica.com

Este otro gráfico nos indica que la cantidad de páginas visitadas en las computadoras es casi el doble que en los smartphones.

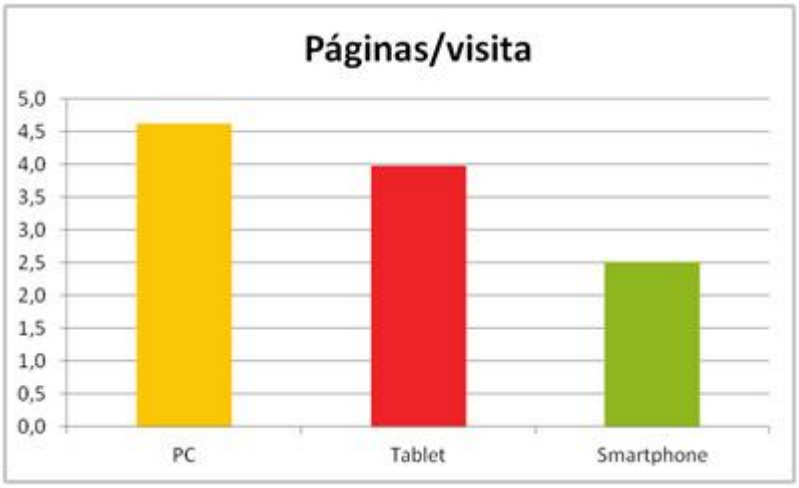

Figura 4. Páginas visitadas según dispositivo
Fuente: http://www.lamagnetica.com

Por todo lo expuesto a lo largo de este capítulo, es vital reducir los tiempos de respuesta de las consultas que se realizan a las Bases de Datos y que las correcciones y/o mejoras que se realizan en el código sean rápidas y fáciles de realizar. Esto puede lograrse aplicando no solamente un estándar de programación sino que también buenas prácticas de programación. Con un código estandarizado se reducen los tiempos de programación y esta resulta más fácil de realizar. Aplicando buenas prácticas se logra aprovechar de manera más eficiente los recursos de Sistema de Gestión de Base de Datos y por ende se reducen los tiempos de respuesta.

4. BUENAS PRÁCTICAS DE PROGRAMACIÓN EN PL/SQL

Acorde al capítulo 1 Fundamentos donde se explicaron las razones por las cuales se cree conveniente la aplicación de buenas prácticas, en este capítulo se detallan las buenas prácticas deberán aplicarse al programar en Oracle PL/SQL.

4.1. OBJETIVO

Este capítulo tiene como objetivo la descripción de recomendaciones a tener en cuenta en la construcción de programas para las distintas aplicaciones desarrolladas en PL/SQL. Este capítulo contiene buenas prácticas de programación basadas en generalidades aplicables a cualquier sistema a construir y que fueron aplicadas exitosamente en otros sistemas.

Dichas recomendaciones son verificadas en el momento de revisión del código que formará parte del sistema en el ambiente productivo, por lo que es útil su difusión a efectos de incorporarlas al momento de desarrollar programas.

4.2. Estándar de programación en PL/SQL

Es importante codificar con estándares por la forma en que la ingeniería del motor de base de datos (RDBMS) maneja las sentencias internamente ya que cuando un SQL es parseado, la base de datos realiza los siguientes pasos:

1. Chequeo de Sintaxis: ¿El SQL está correctamente estructurado? ¿Los paréntesis se corresponden?
2. Búsqueda en Shared SQL Area: ¿Existe una copia parseada de la sentencia en la shared buffer pool de la SGA (System Global Area)?
3. Búsqueda de Diccionario de Datos: chequeo de vistas, sinónimos, privilegios de seguridad
4. Search Path: optimizador de Reglas o Costos
5. Salvado del Execution Plan: almacenado en la SGA

El factor crítico está dado por el segundo paso del parseo. Aquí es cuando se ve lo importante que es seguir estándares en la programación ya que las sentencias SQL no pueden compartirse en la SGA a menos que sean absolutamente idénticas. Si las sentencias se escriben de manera idéntica permanecerán más tiempo en la SGA y cada vez que se las requiera volver a ejecutar, la bases de datos la buscará directamente allí en vez de realizar nuevamente los pasos de validación descriptos anteriormente.

A pesar de su parecido y que el resultado es el mismo, las siguientes sentencias SQL no son las mismas y por lo tanto no serán compartidos en la SGA y se almacenará una copia de cada ella en esta porción de memoria.

```
SELECT NAME
FROM CUSTOMER
WHERE ID = 212;

SELECT NAME FROM CUSTOMER
WHERE id = 212;
```

```
SELECT NAME
FROM CUSTOMER c
WHERE ID = 212;

SELECT NAME
FROM CUSTOMER cus
WHERE id = 212;
```

Ejemplos de estándares:

- Poner todos los verbos SQL en mayúsculas
- Poner los nombres de tablas y campos en minúsculas
- Comenzar todos los verbos SQL en una línea nueva
- Alinear los verbos a la derecha o izquierda con respecto al primer verbo SQL
- Separar todas las palabras con un solo espacio
- Usar siempre el mismo alias para una tabla determinada
- Usar bind variables en favor de hardcodear (quemar) los valores de las variables

En el próximo capítulo se propone un estándar de programación en Oracle PL/SQL. El mismo fue desarrollado luego de 10 años de haber programado código PL/SQL en los cuales se aprendieron ciertos trucos que se describen a continuación y que forman parte de las buenas prácticas utilizadas a lo largo de todos esos años.

4.3. Bind Variables

Las bind variables permiten definir porciones de una sentencia SQL (valores de datos a ser insertados o claves de búsqueda) como "parámetros" de la misma.

Esto provoca que se ejecute la sentencia "n" veces sin necesidad de re-parsear el SQL cada vez que se procesa.

La alternativa a esta opción es reemplazar las variables por literales. Al ser una sentencia diferente cada vez, la misma no puede ser compartida en la Shared SQL Area de la SGA y hace que la sentencia se parsee cada vez que es ejecutada.

Por lo tanto la utilización de bind variables ayuda al mejor aprovechamiento de la memoria SGA del motor de Oracle, ya que permite se reutilicen porciones de memoria ya ocupadas por un mismo query y se evite el tiempo adicional necesario de re-parseo de la sentencia.
El caso típico donde es imprescindible usar bind variables es en la construcción de SQL Dinámicos.

Ejemplo 1

```
VAR v_depno NUMBER;

EXEC :v_deptno := 10;

SELECT ename
FROM EMPLOYEES
WHERE deptno = :v_deptno; --utilización de bind variable
```

Ejemplo 2

```
--utilización de bind variables en el armado de un query dinámico

IF NVL(TRAMITE, '%') <> '%' THEN
    v_where:= v_where||' AND TRAMITE = :v_tramite';
END IF;
```

```
IF NVL(NUMERO, '%') <> '%' THEN
    v_where := v_where||'AND NUMERO = :v_numero';
END IF;

IF NVL(CODIGO, '%') <> '%' THEN
    v_where := v_where||' AND CODIGO = :v_codigo';
END IF;

v_query := 'SELECT NUMERO,
            FECHA,
            TO_CHAR(FECHA_INICIO, "DD-MON-YY HH24:MI"),
            SOLICITUD,
            PREFIJO,
            TELEFONO,
            NUMERO DESTINO
            FROM TRAMITES_REGISTRADOS_V';
v_query := v_query||v_where;


--apertura del cursor*

v_cursor := DBMS_SQL.OPEN_CURSOR;

DBMS_SQL.PARSE(v_cursor, v_query, DBMS_SQL.V7);


--declaracion de las bind variables

IF NVL(TRAMITE, '%') != '%' THEN
    DBMS_SQL.BIND_VARIABLE(v_cursor, ':v_tramite', TRAMITE);
END IF;

IF NVL(NUMERO, '%') != '%' THEN
    DBMS_SQL.BIND_VARIABLE(v_cursor, ':v_numero', NUMERO);
END IF;

IF NVL(CODIGO, '%') != '%' THEN
    DBMS_SQL.BIND_VARIABLE(v_cursor, ':v_codigo', CODIGO);
END IF;
```

4.4. Accesos a la Base de Datos

Los accesos a la base deben ser la menor cantidad posible, un tráfico de red sobrecargado provoca una consecuente degradación en el tiempo de respuesta de la aplicación y por lo tanto se deben realizar la menor cantidad posible de accesos a la base. Para ello se deben seguir las siguientes recomendaciones.

4.4.1. Cursores explícitos

Los cursores implícitos realizan un segundo fetch por cada sentencia SELECT, ya que chequean que el valor seleccionado sea único.

```
SELECT OBSERVACIONES
INTO v_observaciones
FROM TRAMITES
WHERE TRT_NUMERO = v_numero;
```

Si la sentencia es frecuentemente ejecutada se puede definir un cursor explícitamente utilizando la cláusula DECLARE en el bloque PL/SQL, luego dentro del bloque hacer un OPEN, un FETCH y un CLOSE del cursor.

```
DECLARE

CURSOR c_CURSOR IS
    SELECT OBSERVACIONES
    FROM TRAMITES
    WHERE NUMERO = v_numero;

BEGIN
    OPEN c_CUR;
    FETCH c_CUR INTO ...
    CLOSE c_CUR;
END;
```

Al estar declarado el cursor de forma explícita, se minimizarán la cantidad de FETCH y por ende los accesos a la base y resultado final se obtendrá en menor tiempo y se consumirán menos recursos del sistema.

4.4.2. Integridad Referencial

Si la integridad referencial (Foreign Keys) se incorpora en la definición de las tablas de la aplicación, se evitan accesos a través de la red innecesarios. El desarrollador debe

olvidarse de realizar los chequeos de integridad de forma manual mediante la aplicación que está desarrollando y dejar que Oracle los realice automáticamente. De esta forma la base de datos conoce cómo está construida la arquitectura de las tablas y las relaciones con otras tablas y es preferente que ella asuma la verificación de integridad que delegarle esta tarea al programador.

Una extensión de la integridad referencial automática es la cláusula DELETE CASCADE (borrado en cascada) que permitiría purgar varias tablas mediante un solo llamado a la base de datos. Cuando se realiza un borrado de este estilo y se utilizó la integridad referencial al momento de crear las tablas y sus relaciones, la base de datos con un solo acceso borra la información de todas las tablas que compongan la solución. Si no se hubiera utilizado integridad referencial al crear las tablas y si la solución estuviera compuesta por cinco tablas, se hubieran requerido cinco accesos a la base de datos para purgar los datos de la solución, en cambio con la integridad referencial aplicada se necesita de un solo acceso para depurar la información.

4.4.3. Triggers, Packages, Procedures y Funciones almacenados (Stored Procedures)

Estos programas PL/SQL se almacenan en la base de datos y el código es compartido por todos los usuarios que lo utilizan. Al estar almacenados en la base, cuando un cliente invoca alguno de ellos, todas las sentencias que lo componen se ejecutan como una sola llamada a la base, disminuyendo de esta manera el tráfico en la red. Por lo tanto es preferible que todos estos tipos de Stored Procedures se creen y almacenen en la base a que sean creados en el programa fuente.

4.4.4. Combinación de accesos no relacionados

Al combinar querys con accesos no relacionados se minimiza la cantidad de querys que se ejecutan y por ende la cantidad de acceso a las base de datos.

Ejemplo 1:

```
SELECT CDA_NUMERO
FROM CENTROS_DE_ATENCION
WHERE CDA_NUMERO = 123456;

SELECT USR_NUMERO
FROM USUARIOS
WHERE USR_NUMERO = 123456;

SELECT TRT_NUMERO
FROM TRAMITES
WHERE TRT_NUMERO = 123456;
```

Reemplazar por:

```
SELECT TRT_NUMERO,
    USR_NUMERO,
    CDA_NUMERO
FROM TRAMITES
WHERE TRT_NUMERO = 123456;
```

Ejemplo 2:

```
SELECT COUNT(1), SUM(SALARY)
FROM EMP
WHERE DEPT_NO = 020
AND EMP_NAME LIKE 'SMITH%';
```

```
SELECT COUNT(1), SUM(SALARY)
FROM EMP
WHERE DEPT_NO = 30
AND EMP_NAME LIKE 'SMITH%';
```

Reemplazar por:

```
SELECT COUNT(DECODE(DEPT_NO, 20, 'X', NULL)) d20_c,
    COUNT(DECODE(DEPT_NO, 30, 'X', NULL)) d30_c,
    SUM(DECODE(DEPT_NO, 20, SALARY, NULL)) d30_s,
    SUM(DECODE(DEPT_NO, 30, SALARY, NULL)) d30_s
FROM EMP
WHERE EMP_NAME LIKE SMITH%';
```

Ejemplo 3:

```
SELECT NAME
FROM EMP
WHERE EMP_NO = 1234;

SELECT NAME
FROM DPT
WHERE DPT_NO = 10;

SELECT NAME
FROM CAT
WHERE CAT_TYPE = 'RD';
```

Reemplazar por:

```
SELECT e.NAME,
    d.NAME,
    c.NAME
FROM CAT c,
    DPT d,
    EMP e,
    DUAL x
WHERE NVL('X', x.DUMMY) = NVL('X', e.ROWID(+))
AND NVL('X', x.DUMMY) = NVL('X', d.ROWID(+))
AND NVL('X', x.DUMMY) = NVL('X', c.ROWID(+))
AND e.EMP_NO(+) = 1234
AND d.DEPT_NO(+) = 10
AND c.CAT_TYPE(+) = 'RD';
```

Si bien la programación del query puede resultar un poco más compleja es mucho más lo que se gana en tiempos de ejecución que lo se dedica al armado del query. Como se ve en el ejemplo 1, para obtener el resultado final se necesitaban tres querys y por ende tres accesos a la base de datos, mientras que al combinarlos se requirió de un solo acceso para retornar la información.

4.5. Trace de Transacciones

Existe un producto provisto por Oracle para la identificación y seguimiento de los procesos que se ejecutan en la base de datos. Este toma información de los mismos a través de una tabla donde se almacenan datos como tipo de proceso, nombre del proceso, etc. Para ello es necesario que cada vez que se crea un paquete o proceso de base nuevo, se agreguen unas sentencias provistas por las librerías Oracle que sirven para mantener informadas estas tablas que serán consultadas por el Top Session de Oracle que mostrará la información pertinente que permita encontrar las sesiones responsables de acaparar la mayor cantidad de recursos.

Para realizar el trace de transacciones existen el siguiente package provisto por Oracle y sus procedures:

DBMS_APPLICATION_INFO.SET_MODULE (module, action) = setea el nombre del módulo y opcionalmente el de la acción. Debe ser llamado al comienzo del módulo.

DBMS_APPLICATION_INFO.SET_ACTION (action) = sirve para registrar la acción realizada por el modulo. El procedimiento debe ser llamado cada vez que se realiza una nueva transacción y una vez finalizada la acción debe ser puesta en null.

Module: es el nombre del módulo y cuenta con 48 caracteres.

Action: es el nombre, o descripción de la acción y es de 32 caracteres.

Ejemplo:

```
CREATE PROCEDURE PR_ADD_EMPLOYEE( p_name VARCHAR2(20),
                                  p_salary NUMBER(7,2),
                                  p_manager NUMBER,
                                  p_title VARCHAR2(20),
                                  p_commission NUMBER(7,2),
                                  p_department NUMBER(2)) AS

BEGIN
   DBMS_APPLICATION_INFO.SET_MODULE('PR_ADD_EMPLOYEE' ,'INSERT INTO EMP');

   --transacción

   DBMS_APPLICATION_INFO.SET_ACTION(NULL);
END;
```

4.6. Afinación de Sentencias SQL

4.6.1. Elección de la Driving Table (Tabla Pivote)

Si se tiene más de una tabla referenciada en el FROM de la sentencia, debe elegirse cuidadosamente la driving table ya que esta es la tabla que es procesada primero en la ejecución de la sentencia. Si el ranking obtenido luego del WHERE es igual para todas las tablas involucradas, el optimizador de Oracle elige la última tabla nombrada en el FROM como la driving table. Una buena política sería elegir como driving table la de menos

registros.

Ejemplo:

TABLA1 16.384 filas

TABLA2 100 filas

```
SELECT COUNT(1)
FROM TABLA1, TABLA2;
--00.96 segundos

SELECT COUNT(1)
FROM TABLA2, TABLA1;
--26.09 segundos
```

La diferencia es porque Oracle hace un scan y sort de la dirving table y luego hace un scan de la otra tabla con un merge de todas las filas devueltas con las de la driving table. Cuando se realiza un join entre 3 o más tablas, se debe intentar que la tabla dirving sea la de la intersección.

Elegir la driving table correcta reducirá los tiempos de respuesta y optimizará el uso de los recursos del sistema.

4.6.2. Uso eficiente del Where no indexado

Oracle evalúa ecuaciones no indexadas, unidas por el verbo AND desde atrás hacia delante. Esto significa que la última cláusula en la lista del AND es evaluada primero. Por lo tanto

en la secuencia del WHERE hay que tratar de posicionar primero la cláusula AND menos costosa y por último la más costosa que por lo general es la que realiza el mayor filtro en los datos que serán retornados. Con el verbo OR el tratamiento es inverso. Se evalúa primero la cláusula nombrada en primer lugar en la lista de los OR.

Ejemplo:

```
SELECT *
FROM EMP e
WHERE 25 < ( SELECT COUNT(1) FROM EMP
            WHERE EMP_MGR = e.EMP _NO)
OR (EMP_TYPE = 'MANAGER' AND EMP_SALARY > 50000);
```

Reemplazar por:

```
SELECT *
FROM EMP e
WHERE (EMP_SALARY> 50000 AND EMP_TYPE = 'MANAGER')
OR 25 < ( SELECT COUNT(1)
            FROM emp
            WHERE EMP_MGR = e.EMP_NO);
```

4.6.3. Sentencias Joins

Una sentencia JOIN es una sentencia SELECT con más de una tabla en la cláusula FROM. Oracle sólo hace join sobre dos tablas a un mismo tiempo. Si el join es entre más de dos tablas, el procedimiento es hacer join con dos de las tablas y el resultado es joinearlo con la siguiente hasta terminar con todas.

Terminología usada en Joins

```
SELECT class.START_DATE,
    class.INSTR_ID
FROM CLASSES class,
    COURSES course
WHERE class.CRS_ID = course.CRS_ID
-- predicado JOIN (referencia dos tablas)
AND class.STATUS = 'CONF' -- predicado no join (referencia una tabla)
AND course.SHORT_NAME = 'SQLTUN80'; -- predicado single-row (clave única)
```

4.6.3.1. Nested Loops Joins

Una de las tablas se define como OUTER table (o DRIVING table) y la otra es llamada como INNER table. Por cada fila de la OUTER table son devueltas todas las filas de la INNER table que satisfacen el predicado join.

NESTED LOOPS

```
    TABLE ACCESS (...) OF OUTER TABLE
    TABLE ACCESS (...) OF INNER TABLE
```

Se llama index nested loops cuando se usa un índice para acceder a la inner table. Este tipo de operación es conveniente cuando obtengo una pequeña cantidad de filas de la outer table, ya sea porque la tabla es chica o porque se filtran filas a través de predicados no-join. De esta manera la cantidad de veces que accedo a la INNER table es limitada.

4.6.3.2. Sort/Merge Joins

Las filas de cada tabla son sorteadas por la columna del predicado join. Luego se hace un merge sobre los dos conjuntos de filas para finalmente obtener el resultado.

```
MERGE (JOIN)
    SORT (JOIN)
        TABLE ACCESS (...) OF TABLE A
    SORT (JOIN)
        TABLE ACCESS (...) OF TABLE B
```

Generalmente las operaciones de este tipo de joins aparecen con full table scan sobre las dos tablas, pero el acceso puede basarse en un index scan si hay predicados no-join que usan los índices.

Los joins sort/merge tienden a ser más performantes que los join index nested loops si la cantidad de filas que satisfacen la condición del join representan la mayor parte del número total de filas de las dos tablas.

El hash join suele utilizarse cuando es adecuado usar un sort/marge join y una de las tablas es mucho más pequeña que la otra.

4.6.3.3. Sort/Merge Joins versus EXISTS

En general, considerar *joinear tablas* cuando el porcentaje de filas retomadas desde la driving table (por ejemplo el número de filas necesarias para ser validadas contra el subquery del EXISTS) es alto.

Ejemplo 1:

```
SELECT EMP_NAME
FROM EMP e
WHERE EXISTS (SELECT 'X'
                FROM dept
                WHERE DEPT_NO = e.DEPT_NO
                AND DEPT_CA = 'A');
```

Reemplazar por:

```
SELECT EMP_NAME
FROM DEPT d,
    EMP e
WHERE e.DEPT_NO = d.DEPT_HO
AND d.DEPT_CAT = 'A';
```

Por el contrario, se debe elegir el exists cuando el número de filas retomadas desde la driving table es bajo.

Ejemplo 2:

```
SELECT *
FROM DEPT d,
    EMP e
WHERE e.DEPT_NO = d.DEPT_NO
AND e.emp_type = 'MANAGER'
AND d.dept_cat = 'A';
```

Reemplazar por:

```
SELECT *
```

```
FROM EMP e
WHERE EXISTS (SELECT 'X' FROM DEPT
                WHERE DEPT_NO = e.DEPT_NO
                AND DEPT_CAT = 'A')
AND e.EMP_TYPE = 'MANAGER';
```

4.7. Sintaxis Alternativas

4.7.1. UNION ALL en lugar de UNION

La cláusula UNION fuerza que todas las filas retornadas por cada porción del UNION hagan un sort y un merge, y que los duplicados sean filtrados y eliminados antes de que la primera fila sea retornada al usuario. En cambio, un UNION ALL simplemente retoma todas las filas incluyendo duplicados y no realiza sort, ni merge o filtro alguno. Si se sabe que las tablas sobre las que se va a realizar el UNION son mutuamente excluyentes, es mucho más performante usar el UNION ALL ya que de esta manera se evitan accesos y comparaciones que por lo general son innecesarias.

4.7.2. IN o UNION en lugar de OR

Si se realiza un OR sobre columnas indexadas, el optimizador realiza un full table scan en vez de usar el índice.

Ejemplo 1:

```
SELECT *
FROM LOCATION
WHERE LOC_ID = 10
OR REGION = 'MELBOURNE';
```

Reemplazar por:

```
SELECT *
FROM LOCATION
WHERE LOC_ID = 10
UNION
SELECT *
FROM LOCATION
WHERE REGION = 'MELBOURNE';
```

Ejemplo 2:

```
SELECT *
FROM LOCATION
WHERE LOC_ID = 1O
OR LOC_ID = 20
OR LOC_ID = 30;
```

Reemplazar por:

```
SELECT *
FROM LOCATION
WHERE LOC_ID IN (10, 20, 30);
```

Si igualmente se quiere usar el OR tener en cuenta de poner la restricción más selectiva primero en la cláusula WHERE.

4.7.3. NOT EXISTS en lugar de NOT IN

Ejemplo:

```
SELECT NAME, DEPTO
FROM DEPT
WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP);
--devuelve los departamentos que no tengan empleados
```

El plan de ejecución de esta sentencia sería el siguiente:

TABLE ACCESS FULL (DEPT)
TABLE ACCESS FULL (EMP)

Ejecuta un full scan de la tabla EMP, por cada fila recuperada. Asumimos que hay un índice definido sobre la tabla EMP y por lo tanto podemos reescribir la consulta para que use ese índice.

```
SELECT NAME, DEPTO
FROM DEPT
WHERE NOT EXISTS (SELECT DEPTNO
                  FROM EMP
                  WHERE DEPT.DEPTNO = EMP.DEPTNO);
```

TABLE ACCESS FULL (DEPT)
INDEX RANGE SCAN (DEPTNO_IDX)

Por lo tanto es conveniente reescribir el NOT IN con NOT EXISTS ya que de esta manera no realizará un FULL SCAN a todas las tablas.

4.7.4. WHERE en lugar de HAVING

La cláusula HAVING filtra las filas seleccionadas sólo después de que todas las filas fueron fetcheadas. Esto puede incluir sorting, summing, etc.

En cambio el WHERE a medida que hace el fetch filtra las filas que necesita y luego realiza el sort o lo que sea necesario sobre el resultado de la consulta.

Ejemplo:

```
SELECT REGION, AVG(LOC_SIZE)
FROM LOCATION GROUP BY REGION
HAVING REGION <> 'SIDNEY'
AND REGION <> 'PERTH';
```

Reemplazar por:

```
SELECT REGION, AVG(LOC_SIZE)
FROM LOCATION
WHERE REGION <> 'SIDNEY'
AND REGION <> 'PERTH'
GROUP BY REGION;
```

4.7.5. Minimizar cantidad de SubQuerys

Cuando una consulta tiene varios subquerys se requerirán más accesos a la base para retornar los resultados y por lo tanto se pueden reescribir las consultas para combinar o minimizar la cantidad de subquerys.

Ejemplo:

```
UPDATE EMP
SET  EMP_CAT = (SELECT MAX(CATEGORY) FROM EMP_CATEGORIES),
    SAL_RANGE = (SELECT MAX(SALJANGE) FROM EMP_CATEGORIES) WHERE EMP_DEPT = 20;
```

Reemplazar por:

```
UPDATE EMP
SET (EMP_CAT, SAL_RANGE) = (SELECT MAX(CATEGORY), MAX(SAL_RANGE)
                            FROM EMP_CATEGORIES)
WHERE EMP_DEPTO = 20;
```

4.7.6. Usar COUNT(1) en vez de COUNT(*)

Supongamos que una tabla tiene veinte columnas y necesitamos saber la cantidad total de registros de esa tabla. Si se utiliza COUNT(*) el motor de la base de datos Oracle lo que hace es contar la cantidad de registros de las veinte columnas y retornar el resultado, en cambio al utilizar COUNT(1) se le está indicando a la base que solamente cuente la cantidad de registros de la primer columna y esto es mucho más rápido que contar los registros de las n columnas que una tabla pueda llegar a tener.

Por suerte Oracle se dio cuenta de esto y lo incluyo en sus nuevas versiones y entonces es importante tener en cuenta que en las nuevas versiones de Oracle, de 10g en adelante, esto ya lo realiza de forma nativa el motor de la Base de Datos y que para los motores antiguos de Base Datos se recomienda sustancialmente la aplicación de esta buena práctica.

Ejemplo:

```
SELECT COUNT(*)
FROM EMP;
```

Reemplazar por:

```
SELECT COUNT(1)
FROM EMP;
```

4.7.7. Eficiencia en comparaciones

Al realizar comparaciones privilegiar el uso de mayor “>” y menor “<” estricto sumando el inmediato anterior o posterior según corresponda, en vez de mayor o igual “>=” y menor e igual “=<”. Al usarse el igual la base de datos evalúa cada sentencia dos veces tanto por mayor o por menor como por igual. En cambio al utilizar el mayor o menor estricto la sentencia es evaluada una sola vez.

Ejemplo:

```
SELECT *
FROM EMP
WHERE EMP_DEPTO >= 1
```

Reemplazar por:

```
SELECT *
FROM EMP
WHERE EMP_DEPTO > 0
--uso del mayor estricto
```

4.8. Utilización de Índices

La utilización de índices es muy importante para la performance, la misma puede ser vital si es bien utilizada y fatal en caso contrario.

Es importante aclarar que estas son guías generales, y que siempre cada caso debe ser analizado individualmente con respecto al impacto en el sistema.

Hay que tener en cuenta que los índices mejoran la performance de un query, si este selecciona un porcentaje pequeño de filas (< 25%). Por otro lado se debe considerar que la presencia de un índice implica que éste sea actualizado con cada operación de INSERT, UPDATE y DELETE. Lo ideal es comparar la ejecución de una sentencia SQL con y sin índices y ver cuál es más beneficioso.

4.8.1. Uso del ROWID cuando sea posible

Antes que cualquier índice hay que considerar la búsqueda por ROWID ya que es el más rápido de todos los accesos posibles a una tabla. El ROWID es el identificador unívoco de un registro en la base de datos, mientras que un índice es el identificador unívoco de un registro en una tabla.

Ejemplo:

```
SELECT ROWID
INTO emp_rowid
FROM EMP
WHERE EMP_NO = 56722 FOR UPDATE;

UPDATE EMP
SET NAME = 'PEPE'
```

```
WHERE ROWID = emp_rowid;
```

4.8.2. Índices Simples

Los índices simples son aquellos que se definen sobre sólo una columna de una tabla.

Para definir un índice simple, considerar como columnas apropiadas aquellas que se caractericen por ser:

- columnas usadas frecuentemente en cláusulas WHERE
- columnas usadas frecuentemente en JOINS
- columnas con buena selectividad (por ejemplo: porcentaje de filas en una tabla que tienen el mismo valor en la columna indexada). Una selectividad buena es equivalente a un porcentaje bajo. (ejemplo: índices sobre claves primarias son los más selectivos)

$$\%\ \text{selectividad de un índice} = \frac{\#\ \text{filas (tabla)}}{\#\ \text{distintos valores indexados}}$$

Por el contrario, a la hora de crear índices es preferible no hacerlo sobre los siguientes tipos de columnas:

- No indexar columnas con pocos valores distintos, a no ser que aquellos valores para la columna que son seleccionados con más frecuencia, aparecen menos que los otros.

- No indexar columnas que se modifican frecuentemente ya que esto implica actualizar el índice de la tabla cada vez que se actualiza un valor en la tabla.
- No indexar columnas que sólo aparecen en cláusulas WHERE como argumentos de funciones (AVG) u operadores.
- No crear índices sobre tablas pequeñas (8 bloques) ya que si la tabla es pequeña es más rápido realizar un FULL SCAN que una consulta indexada

4.8.3. Índices Compuestos

Son los índices definidos sobre más de una columna.

Ventajas en algunos casos sobre índices simples:

- Mejor selectividad: a veces 2 columnas con mala selectividad pueden combinarse en un índice compuesto con buena selectividad.
- Se evita el acceso a la tabla: si todas las columnas seleccionadas están en la definición del índice, Oracle las recupera desde el índice y no necesita acceder a la tabla.

Una sentencia SQL usa un índice compuesto, si se referencia en la cláusula WHERE un prefijo del índice (leading portion), por ejemplo para la sentencia siguiente:

```
CREATE INDEX IX ON TABLA (X, Y, Z)
```

X, XY, XYZ son prefijos del índice.

Las columnas candidatas a indexar mediante índices compuestos son:

- Columnas usadas frecuentemente y juntas en la cláusula WHERE con operadores AND. Si de estas columnas hay algunas que se usan aún más frecuentemente, crear el índice de tal manera que estas formen un prefijo para el índice.
- Crear el índice ordenando las columnas por selectividad en la sentencia CREATE INDEX.

4.8.4. Anulación de índices

Hay que evitar las cláusulas que anulan el uso del índice innecesariamente. Usualmente se mata un índice con el uso de una función en el campo índice nombrado en el WHERE.

Ejemplo 1:

```
SELECT AMOUT FROM TRANSACTION
WHERE SUBSTR(ACCOUNT_NAME, 1, 7) = 'CAPITAL';
```

Reemplazar por:

```
SELECT AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME LIKE 'CAPITAL%';
```

Ejemplo 2:

```
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE AMOUNT != 0;
```

Reemplazar por:

```
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE AMOUNT > 0;
```

Ejemplo 3:

```
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE TRUNC(TRANS_DATE) = TRUNC(SYSDATE);
```

Reemplazar por:

```
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE TRANS_DATE BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) + 0.99999;
```

Ejemplo 4:

```
SELECT AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME||ACCOUNT_TYPE = 'AMEXA';
```

Reemplazar por:

```
SELECT AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME = 'AMEX'
AND ACCOUNT_TYPE = 'A';
```

Hay tener cuidado con las conversiones de tipo automáticas que hace Oracle al comparar campos de diferente tipos de datos.

```
WHERE MOD_NUMERO = MNU_CODIGO
```

Donde MOD_NUMERO es numérico y MNU_CODIGO es alfanumérico. Oracle hace la siguiente conversión de forma automática:

```
WHERE MOD_NUMERO = TO_NUMBER(MNU_CODIGO)
```

```
WHERE MNU_CODIGO LIKE 10%
```

Oracle lo trasforma en:

```
WHERE TO_CHAR(MNU_CODIGO) LIKE '10%'
```

Según el método de optimización que usa Oracle, si un índice es accesible, el optimizador lo usa. Pero sin embargo hay casos donde no es muy conveniente usar un índice, entonces hay que indicarle al optimizador que lo ignore. Para ello debemos decirle a la base que anule el índice. Los ejemplos anteriormente citados para evitar la anulación de índices hay que verlos de manera inversa.

Ejemplo:

(1)

```
SELECT *
FROM TABLA1
WHERE COLUMNA1 = 'B'
```

(2)

```
SELECT *
FROM TABLA1
WHERE COLUMNA1 = 'A'
```

Condiciones:

valores de COLUMNA1 → [A, Z]

filas TABLA1 = 1000

75% filas = "A"

1 % filas TABLA1 = [B, Z]

índice (COLUMNA1)

(1) se ejecutará más rápido con índice

(2) se ejecutará más rápido con Full Scan sobre TABLA1

El optimizador no tiene la información sobre la distribución de valores para la columna COLUMNA1 y como tiene definido un índice, lo usa.

Habrá que escribir (2) de manera tal que no use el índice:

```
SELECT *
FROM TABLA1
WHERE COLUMNA1||''='A'
```

Si COLUMNA1 fuera de tipo numérico o fecha, baste sumarle un 0 para anular el índice y no alterar el resultado.

4.8.5. Acceso indexado VS full table scan

Oracle recomienda que las tablas que están compuestas por menos de 8 bloques de datos, sean accedidas mediante un full table scan. Para tablas grandes, una búsqueda indexada usualmente es más eficiente. Igualmente esto depende de la cantidad de filas que retome el query. Si el volumen de filas es grande seguramente convendrá hacer un full table scan en vez de un acceso indexado. Ya que si hiciéramos el acceso en forma indexada, por cada fila se debe leer el bloque de índices y el bloque de datos, en cambio con un full table scan solo se lee el bloque de datos. Otra cosa contra el acceso indexado es que se va leyendo en el orden del índice y no en el orden en que fueron ingresados los datos, esto provoca una mayor lectura de bloques físicos pues las claves pueden estar desparramadas por cualquier parte en el disco. En el caso de un full table scan los datos se van leyendo secuencialmente en el orden en que están en los bloques, por lo que no hay posibilidad que se levante un bloque más de una vez a memoria, ni de que la cabeza lectora del disco se mueva demasiado. Lo que no está firmemente establecido es que porcentaje de filas del total de la tabla se considera adecuado para hacer un full table scan.

Esta línea divisoria depende mucho de los parámetros de la base configurados por el DBA como por ejemplo: DB_BLOCK_SIZE, DB_FILE_MULTIBLOCK_READ_COUNT, SHARED_POOL _ SIZE, etc… Generalmente cuando mayor es el tamaño del bloque de la

base menor es el porcentaje de filas necesario para que convenga hacer un full table scan. Podríamos decir que el límite inferior de este porcentaje oscila entre un 20% y un 50% de la tabla.

4.9. Búsquedas de muchos elementos

A la hora de hacer búsquedas es muy común cuando se buscan muchos elementos y los mismos no se encuentran en una tabla, ver querys mal escritos que contienen una larga lista de elementos a ser buscados:

```
WHERE CAMPO IN (ELEMENTO 1, ELEMENTO 2, ELEMENTO 3, ELEMENTO 4, ELEMENTO
4, ELEMENTO 5, ELEMENTO 6, ELEMENTO 7, ELEMENTO 8, ELEMENTO 9, ELEMENTO
10, ELEMENTO 11, ELEMENTO 12, ELEMENTO 13, ELEMENTO 14, ELEMENTO N)
```

Se obtiene una mayor performance si dicha larga lista de elementos se cargan en una tabla y al realizar la búsqueda se hace join con dicha tabla. Hay que tener cuidado de no llenar la base de datos con tablas que van a ser utilizadas pocas veces y que ocuparán espacio innecesario. Por lo tanto si la lista se utilizará para unas pocas búsquedas debe reutilizarse la tabla en la que se cargará dicha lista.

4.10. Uso de declaraciones ancladas

Se puede usar los atributos de declaración %TYPE y %ROWTYPE para anclar el tipo de datos de una variable a la de una variable o estructura de datos previamente existente. La estructura de datos de anclaje puede ser una columna en una tabla de base de datos, toda la

tabla en sí, un registro definido por el programador o una variable PL/SQL local. En el ejemplo siguiente se declara una variable local con la misma estructura que el nombre de la empresa:

```
my_company company.name%TYPE
```

En este segundo ejemplo se declara un registro basado en la estructura de un cursor:

```
CURSOR company_cur IS

SELECT company_id, nombre, incorp_date
FROM company

company_rec company_cur%ROWTYPE
```

Los tipos anclados ofrecen los siguientes beneficios:

Sincronización con columnas de la base de datos. Muchas variables PL/SQL representan la información de la base de datos dentro del programa. Usando % TYPE, se asegura que la estructura de datos de la variable local coincide con la de la base de datos. Si en cambio se codificó con una declaración propia, el programa podría estar fuera de sincronía con el diccionario de datos y generar errores. Por ejemplo, si en un declaración anterior se declaró my_company como VARCHAR2(30) y luego se expandió el tamaño de columna en la tabla a 60, es probable que el programa cause excepciones del tipo VALUE_ERROR.

Normalización de las variables locales. Utiliza variables PL/SQL para almacenar los valores calculados utilizados en toda la aplicación. Se puede usar %TYPE para basar todas las declaraciones de estas variables en un único tipo de datos de variable centralizada. Si

necesita modificar ese tipo de datos, tal vez para expandir el tamaño máximo de un número para reflejar mayores ingresos, sólo se tendrá que cambiar esa declaración única. Todas las declaraciones ancladas recogerán entonces la nueva restricción cuando los programas sean recompilados.

4.11. Uso de Vistas Materializadas

En las vistas materializadas aparte de almacenar la definición de la vista propiamente dicha, Oracle también almacena los registros que resultan de la sentencia SELECT que define la vista. Como en las vistas normales la sentencia SELECT es la base de la vista, pero la sentencia SQL se ejecuta cuando se crea la vista y los resultados se almacenan físicamente como si fuera una tabla real que ocupa espacio en el disco rígido. Las vistas materializadas pueden definirse utilizando parámetros de almacenamiento que se utilizan para una tabla normal (tablespace, increase, etc.). Las vistas materializadas también permiten índices por lo que esta funcionalidad resulta muy útil a la hora de mejorar el rendimiento de sentencias PL/SQL o SQL que utilicen las vistas.

Si una vista utiliza muchas tablas enlazadas (join) de forma compleja y dicha vista va a ser utilizada frecuentemente, será muy conveniente definirla como una vista materializada, esto contribuirá e mejorar el rendimiento de la Base de Datos ya que la sentencia SQL base de la vista sólo se ejecutará una única vez.

4.12. Uso del Bulk Collect

Desde una perspectiva de rendimiento buscar aplicar BULK COLLECT, que ofrecen mejoras en el rendimiento en prácticamente todas las circunstancias ya que facilita la selección masiva de información de la base a alta velocidad, permitiendo el fetch de múltiples filas en una o más colecciones.

Adicionalmente, el uso de FORALL facilita la ejecución masiva de inserts, updates y deletes, permitiendo transferir la información de una o más colecciones a la base. Ambas alternativas pueden ser utilizadas de manera conjunta o individual, y la principal implicancia es el incremento en la performance de las operaciones. La ventaja de BULK COLLECT con respecto a los cursores, radica en que al utilizarse tablas en memoria estas pueden ser consultadas y llenadas sin posiciones fijas.

Se debe utilizar un LIMIT adecuado para cada fetch, ya que utilizar uno extremadamente grande puede generar problemas de memoria en el servidor y podrá afectar a otros procesos que se están ejecutando en la base de datos, y al utilizar uno demasiado chico no se aprovecha la cantidad de procesamiento masivo que esta funcionalidad permite. Según mi experiencia lo ideal es definirlo en un valor entre 100 a 500, personalmente y según el análisis de cada caso elijo el valor 100 porque en base a mi experiencia suele ser el mejor valor. Elijo un valor tan chico y no 1.000 o 5.000 o 10.000, por la simple razón que manejar gran cantidad de datos en memoria es más costoso que manejar poca cantidad.

4.13 Evitar salidas no estructuradas de bucles (loops)

Se debe seguir estas pautas para terminar un bucle:

- Dejar que un bucle FOR complete su número especificado de iteraciones. No usar una sentencia EXIT para salir prematuramente.
- Siempre asegurarse de incluir una instrucción EXIT dentro de un bucle simple.
- Nunca utilizar EXIT para dejar un bucle WHILE.
- No emitir RETURN directamente desde dentro de un bucle

4.14. Codificar un único RETURN para la ejecución de una función exitosa

El único propósito de una función debe ser devolver un valor único, esto podría ser una tabla PL/SQL u otra estructura compuesta. Para reforzar esta individualidad, se debe codificar la función de modo que tenga sólo una declaración RETURN. Este RETURN también debe ser la última instrucción en la función. Utilizo el siguiente ejemplo para lograr ese efecto:

```
FUNCTION fname() RETURN datatype IS
RETURN_value datatype;
BEGIN
RETURN return_value;
END fname
```

4.15. Uso de SELECT *

Nunca usar SELECT * ya que en algún momento modificará la tabla o vista que se está consultando. Si se ha eliminado la columna que se necesita la aplicación terminará mostrando un error. Si se han agregado varias columnas, la ejecución de la consulta tomará

más de lo que realmente se necesita, y esto genera gastos innecesarios en la base de datos y tráfico de red adicional.

4.16. Uso de Restricciones (Constraints)

Las restricciones son imprescindibles para las bases de datos. Sin embargo, cuando se agrega restricciones NOT NULL a columnas que pueden tener datos faltantes (NULL), fuerza a los usuarios a ingresar basura.

Una de las herramientas que se pueden utilizar para rechazar valores de datos erróneos que entran en tablas son las restricciones CHECK. Este tipo de restricción está destinado a ser utilizado para validar una sola fila basada en un predicado que es una función de cualquier combinación de las columnas de la tabla en esa fila. La diferencia entre usa ninguna restricción y un pequeño número de restricciones simples es mensurable. Una restricción compleja es significativamente más costosa que una simple restricción. Deshabilitar las restricciones CHECK reducirá el tiempo de carga, tal vez significativamente. La sobrecarga de una restricción CHECK se determina por su complejidad de evaluación. La evaluación de las restricciones CHECK tiene un costo fijo y por lo tanto un costo relativo menor cuando las otras partes del proceso son más caras.

4.1.7. Colapso del predicado

El colapso del predicado se produce cuando un predicado de columna implica más de una bind variable. Una expresión de la forma [col = DECODE (: b1, ",: b3, col)] es un ejemplo de colapso predicado. Esto implica que si la bind variable 1 es nula, entonces se debe usar

la bind variable 3. De lo contrario, la expresión resultará en [col = col]. Esto evita que el optimizador utilice el índice en la columna col debido uso de DECODE.

En el ejemplo siguiente se muestra cómo se usa el colapso de predicado para contraer una bind variable de nombre con la bind variable delivery_id en un solo filtro. Al realizar un EXPLAIN PLAN, esto resulta en un full table scan.

```
SELECT delivery_id, planned_departure_id, organization_id, status_code
FROM wsh_deliveries
WHERE delivery_id = NVL(:b1,delivery_id) AND name = DECODE(:b1,'',:b3,
NAME)
ORDER BY UPPER(HRE.full_name)
```

Esta consulta se puede volver a escribir utilizando un UNION para cortocircuitar un solo lado del UNION basado en los valores de la bind variable. Por ejemplo, si se proporciona el enlace delivery_id, sólo se ejecuta la primera rama de UNION. Si se suministra un valor para la bind variable nombre, se ejecuta la segunda rama de UNION. En ambos casos, ambos lados de la UNION utilizan índices bastante selectivos en la columna delivery_id o en la columna name. Esto es mucho más eficiente que la consulta original que realizó un full table scan.

```
SELECT delivery_id, planned_departure_id, organization_id, status_code
FROM wsh_deliveries
WHERE delivery_id = :b1 AND (:b1 IS NOT NULL)
UNION
SELECT delivery_id, planned_departure_id, organization_id, status_code
FROM wsh_deliveries
WHERE name = :b2 AND (:b1 is null)
```

4.18. Análisis de estadísticas del optimizador

Las estadísticas del optimizador son una colección de datos que describen más detalles sobre la base de datos y los objetos de la base de datos. Estas estadísticas se almacenan en el diccionario de datos y son utilizadas por el optimizador de consultas para elegir el mejor plan de ejecución para cada sentencia SQL. Las estadísticas del optimizador incluyen lo siguiente:

- Estadísticas de tabla (número de filas, bloques y la longitud promedio de fila)
- Estadísticas de columnas (número de valores distintos en una columna, número de valores nulos en una columna y distribución de datos)
- Estadísticas del índice (número de bloques foliares, niveles y factor de agrupamiento)
- Estadísticas del sistema (rendimiento y utilización de CPU y E / S)

Las estadísticas del optimizador se almacenan en el diccionario de datos. Pueden visualizarse utilizando vistas de diccionario de datos similares a las siguientes:

```
SELECT * FROM DBA_TAB_STATISTICS
```

Debido a que los objetos de una base de datos pueden cambiar constantemente, las estadísticas deben actualizarse regularmente para que describan con precisión estos objetos de base de datos. Las estadísticas se mantienen automáticamente por el motor de la base de datos o pueden mantenerse las estadísticas del optimizador manualmente utilizando el paquete DBMS_STATS.

4.19. Uso de Hints

Los Hints le permiten al programador tomar decisiones que tomaría el optimizador. Como desarrollador de aplicaciones, es posible que tenga información sobre los datos que el optimizador desconoce. Los Hints proporcionan un mecanismo para instruir al optimizador a elegir un plan de ejecución de determinadas consultas basado en criterios específicos. Los Hints son en realidad comentarios en sentencias SQL que son leídas por el optimizador y se indican con el signo más en / * + HINT * /. Sin el signo más el comentario sería sólo eso un comentario. El optimizador no se preocupa por los comentarios que añada para aumentar la calidad o la legibilidad de su código. Con el signo más, el optimizador lo utiliza para determinar el plan de ejecución de la sentencia.

Por ejemplo, es posible que sepa que un determinado índice es más selectivo para determinadas consultas. Sobre la base de esta información, es posible que pueda elegir un plan de ejecución más eficiente que el optimizador. En este caso, utilizar Hints para indicar al optimizador que utilice el plan de ejecución óptimo.

Si se desea ejecutar rápidamente una consulta a la tabla de ventas del año pasado mientras el sistema está inactivo. En este caso, podría construir la siguiente consulta:

```
SELECT /*+ PARALLEL(s,16) */ SUM(amount_sold)
FROM sales s
WHERE s.time_id BETWEEN TO_DATE('01-JAN-2015','DD-MON-YYYY')
AND TO_DATE('31-DEC-2015','DD-MON-YYYY')
```

Otro uso común para Hints es asegurar que los registros se cargan eficientemente mediante compresión. Para ello, utilizar el Hint APPEND como se muestra en el siguiente SQL:

```
INSERT /* +APPEND */ INTO TABLA
```

Usar Hints para:

- Para fines demostrativos.
- Para probar diferentes planes de ejecución en la fase de desarrollo.
- Proporcionar a Oracle la información pertinente cuando dicha información no está disponible en las estadísticas.
- Para arreglar el plan de ejecución cuando esté absolutamente seguro que los Hints conducen a un aumento significativo de rendimiento.

4.20. Particionado de tablas

El particionado es una técnica utilizada para distribuir el almacenamiento de tablas, índices y vistas materializadas en más de estructura de almacenamiento que se administren de forma independiente. Esta división de almacenamiento es lo que se llama particionado. Como dice el dicho "divide y vencerás", pues eso es lo que se logra con el particionado, logrando mejorar el rendimiento, disponibilidad y mejor manejabilidad de los datos en la base de datos.

El particionado se lo realiza utilizando una clave de particionado (Partitioning Key) que es la que identifica en que parte de la partición se almacenará el dato. Hay varios criterios a considerar para determinar si se necesita particionar una tabla:

- El tamaño de la tabla supera los 2G de tamaño.

- La tabla contiene información histórica que tiende a ser modificada, un ejemplo es en la facturación donde se genera mucha información por año y mucha de esta información se mantiene para análisis de los históricos, dicha información podríamos dividirse en dos particiones una histórica y otra actual.

Oracle ofrece tres técnicas básicas de particionado y es totalmente transparente para las aplicaciones que accedan a los datos.

Range Partitioning, esta técnica usa un rango de valores para determinar en qué partición de datos se almacenará el registro. Este es el método más común de particionar los datos y se usa a menudo como clave de partición los campos fechas.

List Partitioning, para esta técnica se define una lista de valores para la clave de particionado, la ventaja de este particionado es que se puede agrupar y organizar los datos que tengan una relación común, como por ejemplo se puede definir como clave de partición el dato PAIS.

Hash Partitioning, esta se basa en un algoritmo hash sobre la columna que se definió como clave de partición, esto es una técnica que distribuye los datos en las particiones de manera equitativa y es usado cuando no se encuentra claros criterios de particionado.

4.21. Purga de particiones

La purga de particiones, que también se conoce como eliminación de partición, se utiliza para tablas particionadas cuando no es necesario acceder a todas las particiones. La purga es visible en el plan de ejecución como números enteros en las columnas PSTART, el

número de la primera partición y PSTOP, el número de la última partición a la que se accede. Si no se ningún número (por ejemplo, KEY), esto significa que Oracle no pudo determinar los números de partición en el momento del análisis, pero piensa que es probable que ocurra una purga dinámica (es decir, durante la ejecución). Esto suele suceder cuando hay un predicado de igualdad en una clave de partición que contiene una función o si hay una condición de combinación en una columna de clave de partición que una vez unida con una tabla particionada. Para las tablas hash particionadas, sólo puede haber purga si el predicado en la columna de clave de partición es un predicado igual o IN list. Si la clave de partición hash implica más de una columna, todas estas columnas deben aparecer en el predicado para que la poda de partición pueda producirse.

4.22. Consultas distribuidas

Las consultas distribuidas acceden a datos desde al menos un origen de datos remoto. Para disminuir las entradas y salidas y así mejorar el rendimiento de la ejecución de una sentencia SQL, se debe minimizar la cantidad de datos que se van a trasladar, especialmente a través del enlace de la base de datos. Con DRIVING_SITE(TABLA) le indica al optimizador que utilice la base de datos en la que TABLA reside como la ubicación para realizar todas las operaciones; Todos los datos que se requieren para ejecutar la sentencia se mueven a esa base de datos a través de enlaces de base de datos que emanan de la base de datos de inicio (local) a los orígenes de datos remotos. Esto puede ser necesario porque la base de datos local puede no tener acceso a las estadísticas de los sitios remotos. Oracle solo elige una base de datos remota sin la indicación DRIVING_SITE cuando todas las fuentes de fila están en ese sitio.

4.23. Optimización de uso de tipos

Al definir los tipos de datos para caracteres, utilizar VARCHAR2 en vez de CHAR. Supongamos que queremos almacenar la palabra hola y en la tabla el campo que se utilizará para guardar este valor es CHAR(6), lo que Oracle hace es utilizar los primeros 4 caracteres para almacenar el dato y rellenar los 2 caracteres restantes con espacios con lo que el dato final en la tabla se guarda como 'hola '. En cambio, si se utiliza VARCHAR2 y el campo de la tabla es VARCHAR2(6) este solamente almacenará el valor exacto del dato a guardar se almacenará 'hola'. Eso optimizará el espacio de almacenamiento en la base de datos y permitirá que las consultas se devuelvan más rápido dado que el valor a recuperar pesará menos y por lo tanto se podrán recuperar de la base datos más bloques con una mayor cantidad de datos.

4.24. Uso de facilidades de rastreo

La base de datos Oracle ofrece una serie de funciones integradas de rastreo que son muy útiles para recopilar datos crudos sobre el rendimiento de un programa. Todo programador debe familiarizarse con los paquetes suministrados y las tablas subyacentes para PL/SQL Profiler, PL/SQL Trace, el Hierarchical Profiler (analizador jerárquico) y PL/SQL Application Data Profiler (analizador de datos de aplicaciones).

El PL/SQL Profiler es accesible a través del paquete DBMS_PROFILER, calcula el tiempo que el programa pasa en cada línea y en cada subprograma. Estas estadísticas de tiempo de ejecución se escriben en un conjunto de tablas de la base de datos. La generación de

perfiles devuelve un número enorme de datos; El aspecto más difícil de trabajar con DBMS_PROFILER es extraer información significativa. La mayoría de los editores PL/SQL ofrecen interfaces gráficas que permiten ayudar a entender los datos del perfil y también facilitan la activación del perfilamiento. No se deben realizar cambios en el código para utilizar este generador de perfiles, sin embargo se tendrá que iniciar y detener el paquete DBMS_PROFILER. Aquí hay un ejemplo del tipo de código que cualquier programador puede escribir en su código para utilizar el perfilador:

```
DECLARE

profile_id PLS_INTEGER;
BEGIN
      DBMS_PROFILER.START_PROFILER(run_comment    =>    '<descriptor    del
perfil>',
                                   run_number => profile_id);
      --codigo de la apicación
      DBMS_PROFILER.STOP_PROFILER ();
END;
```

Oracle ofrece el paquete DBMS_TRACE que permite controlar el seguimiento de la ejecución de subprogramas en las aplicaciones. También se pueden rastrear excepciones, ejecución SQL y ejecución de línea individual. La información se escribe en un conjunto de tablas propiedad de SYS que se crean por el script tracetab.sql ubicado en el directorio <Oracle_home> Rdbms\Admin de la instalación de la base de datos Oracle). No se tendrá que realizar cambios en el código para utilizar este mecanismo de seguimiento.

El PL/SQL Hierarchical Profiler es nuevo en la base de datos Oracle 11 g, está expuesto a través del paquete suministrado DBMS_HPROF y perfila la ejecución de un programa. Almacena los datos de una manera que revela claramente la pila jerárquica de la aplicación. La mejor manera de entender el valor de DBMS_HPROF es contrastarlo con el generador de perfiles original (DBMS_PROFILER). Este rastrea la información hasta la línea

individual, lo que dificulta que los programadores vean el rendimiento de distintas unidades de código (procedimientos y funciones). El nuevo paquete DBMS_HPROF permite ver el rendimiento de estas unidades en un nivel superior. No se tendrá que realizar cambios en el código para utilizar este generador de perfiles.

Application Data Profiler puede utilizarse con el paquete DBMS_APPLICATION_INFO para rastrear la ejecución de programas y/o partes de programas específicos. La salida de este paquete se escribe en una variedad de vistas de V$, lo que significa que se puede revisar y analizar información de rastreo mientras la aplicación sigue en ejecución. Este paquete es especialmente útil en el análisis de programas de larga duración. Se tendrán que hacer cambios en el código para aprovechar este perfilador.

4.25. Optimización en la compilación

A partir de Oracle 10 g, el compilador PL/SQL ahora también reorganizará automáticamente el código para mejorar el rendimiento. En Oracle 10 g, hay tres niveles diferentes de optimización:

Nivel 2: esta es la forma predeterminada y la más agresiva de optimización. Aplica la mayoría de los reajustes a los elementos del código, tiene el mayor potencial para mejorar el rendimiento y también tiene el mayor impacto en el tiempo de compilación ya que lo aumenta.

Nivel 1: el compilador realiza optimizaciones básicas en el código, pero no es tan agresivo como en el Nivel 2. Se verán mejoras de rendimiento limitadas, pero el tiempo de compilación tampoco se incrementará de forma tan dramática.

Nivel 0: la optimización está deshabilitada. El código se compila como lo haría en la versión de Oracle 9 i.

En Oracle 11 g se ofrece un nuevo nivel:

Nivel 3: cuando se establece la optimización del compilador a este nivel, el compilador PL/SQL automáticamente pone en línea todas las llamadas a los subprogramas locales en el programa actual. Es decir, reemplaza la llamada al subprograma con el código que está contenido en ese subprograma, lo que resulta en una optimización adicional.

Por lo tanto, la buena práctica es simple, hay que asegúrese que cuando el código se compila se esté utilizando el nivel de optimización más alto posible. En general, sólo se debe seguir con el valor predeterminado. Sería muy inusual encontrarse con una situación que requiera un nivel de optimización reducida.

4.26. Malas prácticas

Al revisar código a lo largo de los años me he encontrado con los siguientes errores que por el uso reiterado se convierten en malas prácticas que pasan de programador en programador:

- Incorporación de SQL complejo dentro del código PL/SQL
- Mala manipulación de errores PL/SQL
- Quemado el tamaño de las variables PL/SQL
- No utilizar bind variables
- Almacenamiento de ROWID para referencia posterior
- Almacenamiento de un LOB vacío en lugar de NULL
- Uso de COMMIT o ROLLBACK dentro de procedimientos almacenados o funciones
- Uso de números mágicos y cadenas en lugar de NULL
- Usar solamente la excepción WHEN OTHERS
- Uso de funciones no deterministas directamente en condiciones
- Acceso de cliente como propietario de un objeto
- Manejo de errores con parámetros de salida
- Formatear datos en vistas
- Ignorar deliveradamente excepciones WHEN OTHERS THEN NULL
- Confiar en predicados de columnas condicionales en triggers
- Confiar en información de contexto en variables de paquete
- Confiar en la inicialización del contexto externo

- Procesamiento secundario con triggers no blindados
- Control transaccional en procedimientos no autónomos
- Uso de la secuencia nextval sin curval
- Uso de una secuencia como contador
- Uso de valores de columna derivados para comprobaciones de existencia
- Uso de excepciones para el control de flujo

5. ESTÁNDAR DE PROGRAMACIÓN EN PL/SQL

Acorde al capítulo 1 Fundamentos donde se explicaron las razones por las cuales se cree conveniente la aplicación de estándares de programación, en este capítulo se detallan el estándar que deberá aplicarse al programar en Oracle PL/SQL.

5.1. OBJETIVO

El objetivo de este estándar es poder unificar la manera en que se programa PL/SQL para que al leer el código escrito por los programadores, su lectura resulte simple y llevadera. Evitando así retrabajo y por ende pérdida de tiempo.

Aplicando este estándar se logra código compacto y de fácil lectura, de esta manera se eleva el nivel de productividad del programador, se minimiza el retrabajo y se evita la pérdida de tiempo tratando de entender lo realizado por otro programador.

5.2. CONVENCIONES

Aquí se detallan todas las convenciones de carácter **obligatorio** que deberán ser respetadas por **todos** los programadores.

5.2.1. Palabras en mayúscula

A continuación se detallan las palabras que deberán escribirse en mayúscula:

- palabras reservadas del lenguaje
- nombres de tablas
- campos de tablas
- nombres de cursores

5.2.2. Nombres de packages, procedures, funciones

Los packages o paquetes deberán comenzar con el prefijo "**PA_**" y terminar con el sufijo "**_PKG**" y su nombre deberá ser escrito en mayúsculas. El nombre a utilizarse deberá ser representativo y consistente con la funcionalidad solicitada.

Los procedures o procedimientos deberán comenzar con el prefijo "**PR_**" y su nombre deberá ser escrito en mayúsculas. El nombre a utilizarse deberá ser representativo y consistente con la funcionalidad solicitada.

Las function o funciones deberán comenzar con el prefijo "**FU_**" y su nombre deberá ser escrito en mayúsculas. El nombre a utilizarse deberá ser representativo y consistente con la funcionalidad solicitada.

5.2.3. Parámetros de entrada y salida

Tanto los parámetros de entrada (INPUT) como los de salida (OUTPUT) deberán comenzar con el prefijo "**p_**", su nombre deberá ser escrito en minúsculas y luego deberán ser sucedidos por el nombre del parámetro. Los parámetros de tipo tabla PL/SQL deberán empezar con el prefijo "**p_tbl_**", su nombre deberá ser escrito en minúsculas y luego deberán ser sucedidos por el nombre del parámetro. Dicho nombre, para los dos tipos de parámetros, deberá ser representativo y consistente con lo que parámetro representa. No deberán usarse nombres del estilo ***p_fec_nac*** ya que los mismos a veces resultan confusos, en su lugar deberá privilegiarse el uso de nombres completos, como ser ***p_fechaNacimiento*** utilizando la convención del camellado.

5.2.4. Declaración de tipos

Todo tipo de dato definido por el programador deberá ser escrito en minúsculas y deberá respetar la siguiente convención:

Vectores

```
TYPE vec_varchar30 IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

Registros (Record)

```
TYPE r_constraint IS RECORD
 (v_constraintName   VARCHAR2(30),
  v_constraintType   VARCHAR2(1),
  v_tableName        VARCHAR2(30),
  v_owner            VARCHAR2(30),
  v_constraintName   VARCHAR2(30),
  v_status           VARCHAR2(8));
```

```
TYPE vec_constraint IS TABLE OF r_constraint INDEX BY BINARY_INTEGER;
```

5.2.5. Declaración de variables

Toda variable que se declare ya sea en forma global o en forma local deberá comenzar con el prefijo "**v_**" y las variables de tipo tabla PL/SQL deberán empezar con el prefijo "**t_**", ambas deberán ser escritas en minúsculas, y luego deberán ser sucedidas por el nombre de la variable. Dicho nombre deberá ser representativo y consistente con lo que la variable representa. No deberán usarse nombres del estilo ***v_fec_nac*** ya que los mismos a veces resultan confusos, en su lugar deberá privilegiarse el uso de nombres completos, como ser ***v_fechaNacimiento*** utilizando la convención del camellado.

5.2.6. Uso de alias

Todo alias que se utilice en una sentencia SQL deberá ser sencillo de reconocer para que pueda ser fácilmente diferenciado de una tabla, para ello todo alias que sea realice en un SELECT deberá comenzar con el prefijo "**a_**" y luego deberá ser sucedido por un nombre escrito en mayúsculas. Dicho nombre deber ser representativo de la tabla o del campo que se está utilizando. Por ejemplo si la tabla se llama EMPLOYEE el alias bien podría ser a_EMPLEADOS.

```
SELECT a_MOTIVOS.MOTIVO_ID a_MOTIVO_ID,
    a_SUBMOTIVOS.SUBMOTIVO_ID a_SUBMOTIVO_ID,
    a_SUBMOTIVOS.DESCRIPCION_SUBMOTIVO a_DESCRIPCION_SUBMOTIVO
FROM CCT_MOTIVO a_MOTIVOS,
    CCT_SUBMOTIVO a_SUBMOTIVOS
WHERE a_MOTIVOS.MOTIVO_ID = a_SUBMOTIVOS.MOTIVO_ID
```

5.2.7. Capitalización de palabras reservadas y otros objetos

Tanto las palabras reservadas del lenguaje (BEGIN, END, SELECT, FROM, INSERT, DELETE, UPDATE, DROP, COMMIT, ROLLBACK, etc.), como los nombres de tablas, vistas, columnas, nombres de procedimientos, funciones y demás objetos especificados puntalmente en los puntos anteriores de este documento, deberán ser escritos en mayúscula y en el caso de nombres compuestos deberán ser separados con guiones bajos "**_**".

5.2.8. Vistas

Todo nombre de una vista deberá ser escrito en mayúsculas y deberá terminar con el sufijo "**_VW**". El nombre de la vista deberá ser representativo con la funcionalidad solicitada, y en el caso de ser compuesta por una sola tabla deberá agregarse al nombre de la tabla el sufijo "**_VW**".

5.2.9. Triggers

Todo nombre de un trigger deberá ser escrito en mayúsculas y deberá comenzar con el sufijo "**TRG_**" y deberá contener las siguientes abreviaturas acorde a la acción que realice

- AF para after
- BF para before
- INS para insert
- UPD para update
- DEL para delete

- DRP para drop

El nombre del trigger a utilizarse deberá ser representativo y consistente con la funcionalidad solicitada.

5.2.10. Cursores

Deberán empezar con el prefijo "**CUR_**", su nombre deberá ser escrito en mayúsculas y deberán estar sucedidos por el nombre del cursor. Dicho nombre deberá ser representativo y consistente con lo que cursor representa.

5.2.11. Versionado

Todo package, procedure, function, trigger, etc. deberá contar con un versionado, lo cual permite realizar a simple vista un análisis de los cambios realizados. A continuación se adjuntan los encabezados para los distintos versionados.

El siguiente encabezado deberá utilizarse en packages, procedures, funciones, triggers, etc:

```
/*****************************************************************
NOMBRE: PA_xxx_xxx_xxx_PKG

REVISIONES:
Versión    Fecha        Autor             Descripción
--------  -----------  ----------------  ---------------------------------
---
1.0       DD/MM/YYYY   Juan Pérez        creación, requerimiento REQxxx
1.1       DD/MM/YYYY   Carlos Gómez      modificación, requerimiento
REQxxx
*****************************************************************/
```

5.2.12. Documentación de código

Es de suma importancia que el código esté comentado, ya casi ningún software (por no decir ninguno) es mantenido durante toda su vida por el autor original. Los comentarios contribuyen a un mejor entendimiento del código evitando pérdidas de tiempo al intentar entender lo anteriormente realizado por otro programador. Ante cada modificación realizada a un programa ya existente, es necesario que la misma quede documentada de la siguiente manera:

Para comentarios en bloque:

```
/*Juan Pérez DD/MM/YYYY REQxxx + una breve explicación de lo realizado*/
```

Para comentarios en una sola línea:

```
--Juan Pérez DD/MM/YYYY REQxxx + una breve explicación de lo realizado
```

5.2.13. Manejo de excepciones

Todo procedure y todos los querys dentro de los mismos deberán contar al menos con el manejo de la excepción OTHERS. También según el caso, deberá privilegiarse el uso del resto de las excepciones (NO_DATA_FOUND, TOO_MAY_ROWS y excepciones declaradas por el programador). El uso de la excepción TOO_MANY_ROWS es **obligatorio** en el caso de realizar SELECT INTO.

Las excepciones generadas por el programador deberán empezar con el prefijo “e_” y luego deberán ser sucedidas por el nombre de la excepción. Dicho nombre deberá ser

representativo y consistente con lo que excepción representa. No deberán usarse nombres del estilo *e_leer_sig* ya que el mismo a veces resulta confuso, en su lugar deberá privilegiarse el uso de nombres completos, como ser *e_leerSiguiente* utilizando la convención del camellado o CamelCase.

5.2.14. Manejo de transacciones

Una transacción la podemos definir en SQL como una colección de sentencias DML que forman una unidad lógica de trabajo o procesamiento, con unas propiedades bien definidas. Las sentencias DML son los comandos que permiten manipular los datos como ser INSERT, UPDATE y DELETE que son comandos DML. De esta manera se da un conjunto de operaciones sobre los datos en una base de datos que se ejecuta entera o no se ejecuta ninguna de sus sentencias. Por lo tanto deberá hacerse uso correcto de COMMIT, SAVEPOINT y ROLLBACK según sea requerido. Supongamos que una transacción está compuesta por tres operaciones y al realizar la segunda se da un error, con lo cual la primer operación se ejecutó correctamente, la segunda operación dio error al ejecutarse y por lo tanto la tercer operación no se ejecutó, esto causa inconsistencias en la Base de Datos y por lo tanto para que no haya transacciones incompletas se deberá ejecutarse ROLLBACK para deshacer la operación. Si las tres operaciones se ejecutan correctamente se deberá ejecutar COMMIT para confirmar el grabado completo de toda la transacción en la Base de Datos.

5.2.15. Uso de logs

Para poder realizar el seguimiento correspondiente a una transacción u o alguna operación y así poder ver qué pasó con la misma; o en caso de errores determinar rápidamente qué fue lo que sucedió y así poder dar una pronta solución sin perder tiempo buscando pistas de lo ocurrido, se deben guardar logs que indiquen paso a paso las acciones realizadas y el error sucedido. Como los logs ocupan espacio en la base de datos se deben depurar los mismos cada cierto período de tiempo que deberá ser determinado por el DBA según la tasa de crecimiento de la base de datos.

5.2.16. Identación del código

Es de gran utilidad y por lo tanto es obligatorio identar todo el código. Que el código está identando no solo da imagen de prolijidad sino que también hace que su lectura y seguimiento sea mucho más fácil y llevadera y por ende haya menos confusiones y errores al leer el código. Para realizar esto deberá usarse el *TAB* y el mismo deberá estar configurado en, TOAD o en SQL Navigator o la herramienta que utilice cada programador para que utilice 4 espacios.

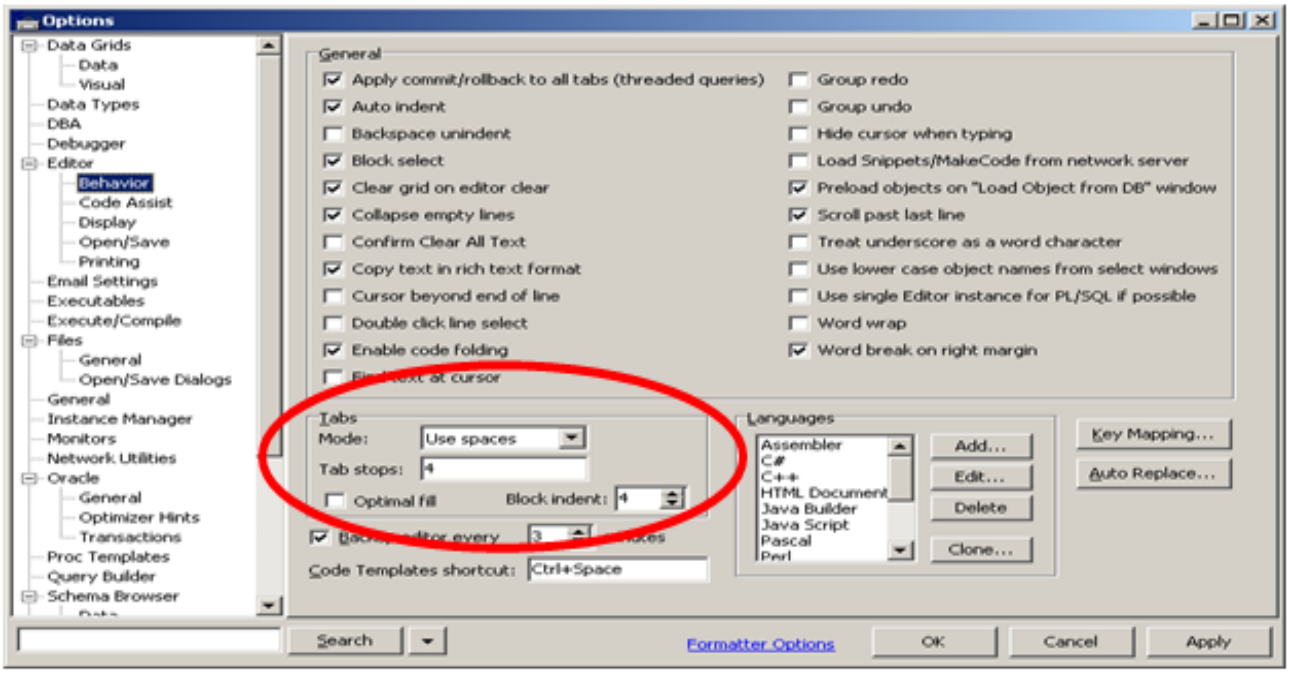

Figura 5. Pantalla de opciones del programa TOAD
Fuente: Programa TOAD

6. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

Si bien uno de los objetivos era medir la concurrencia, medir el acoplamiento y medir la cohesión de código, esto lamentablemente no fue posible dado que ninguna de las cuatro empresas en las cuales se solicitó realizar esta medición (dos empresas de servicios de consultoría informática, un banco y empresa de telecomunicaciones) permitieron realizar dichas mediciones aduciendo razones de seguridad y confidencialidad tanto de su código fuente como de la información de sus clientes. Las empresas solicitaron no se revele su identidad y para el caso se las llama XBE, XTE, XRE y XME.

Inclusive se intentó realizar las mediciones en una base de datos de laboratorio, pero dado que dicha base de datos al ser de pruebas no contaba con el tamaño, estructura, uso permanente y concurrente, código fuente de una aplicación real, confiabilidad y concurrencia tanto de usuarios como de transacciones, por lo tanto las mediciones realizadas arrojaron resultados muy por encima de los esperados. Los cuales luego de varios análisis fueron descartados ya que no representaban la realidad de un ambiente productivo.

La única medición real pero extraoficial que se pudo realizar en un ambiente productivo real fue la corrección de un reporte en la empresa XME cuya ejecución se demoraba 23 minutos en brindar los resultados al usuario final, y luego de aplicar tanto el Estándar de Programación como el Compendio de Buenas Prácticas se logró reducir el tiempo

considerablemente. No se reestructuró el reporte y tampoco se revisó ni analizó su lógica para determinar si lo que se estaba realizando era correcto, pero con tan solo aplicar las recomendaciones de este documento el tiempo final de ejecución se redujo a entre 3 y 5 minutos dependiendo este de la carga con la que la Base de Datos se encontrara al momento de la ejecución. Como se puede notar, el tiempo fue reducido entre un 78% y 87% en promedio podríamos decir que un 83% lo cual va más allá de lo esperado.

¿Cómo se puede explicar este resultado?

Simplemente, el reporte no estaba bien construido ya que no se aprovechaban las capacidades de procesamiento de la Base de Datos y con 2 simples correcciones, por mencionar solamente las que más influyeron, se obtuvo tan inmejorable resultado.

1. Se cambió el uso de SELECT COUNT (*) por el uso de SELECT COUNT (1), lo cual ayudó a que no se contaran todas las columnas de la tabla y por consiguiente al solamente contarse 1 columna se insumió menos tiempo en obtener el resultado que se requería para algunas sub consultas del reporte.
2. Cuando se utilizaba BULK COLLECT para obtener grandes volúmenes de datos, el valor para la cláusula LIMIT era muy chico de 10 registros y se cambió el valor para que procesara en bloques de a 200 registros. Es como si empleara una taza para sacar agua y luego se cambiara la taza por un balde. Por simple lógica con el balde se puede sacar mucha mayor cantidad de agua que con una taza. Eso

contribuyó a que se pudieran procesar al mismo tiempo una mayor cantidad de registros y se redujeran la cantidad de ciclos necesarios para procesar todos los registros requeridos y por ende se redujo considerablemente el tiempo.

Por todo lo expuesto anteriormente, para validar tanto el Compendio de Buenas Prácticas como el Estándar de Programación se recurrió al juicio de expertos a los cuales se les realizó las encuestas que se encuentran en la sección Anexos.

Para determinar el tamaño de la muestra se utilizó una calculadora online provista por la página http://www.netquest.com y como resultado se puede apreciar en la figura 6 que la muestra a evaluar es de 44 personas. Las personas encuestadas pertenecen a las áreas de delivery, fábrica de software, arquitectura y administración de base de datos de una empresa que brinda servicios de consultoría informática y que utiliza para sus labores diarias de programación en Bases de Datos el lenguaje Oracle PL/SQL. La empresa solicitó no se revela su identidad y es por eso que para este estudio se la llamó XTE.

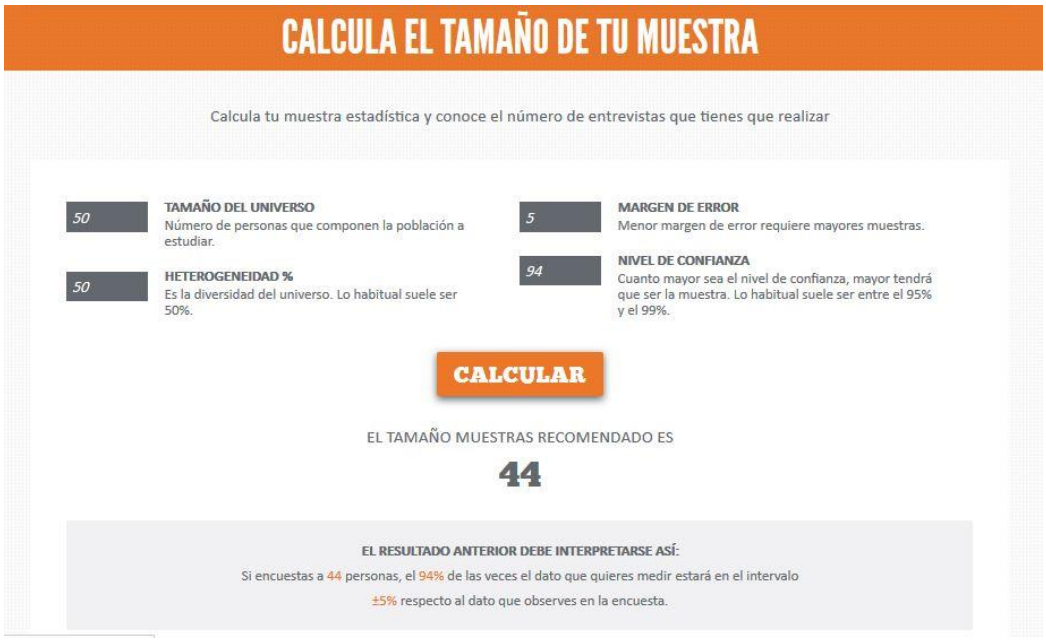

Figura 6. Calculadora de tamaño de la muestra
Fuente: http://www.netquest.com

6.1. Resultados de la encuesta del Compendio de Buenas Prácticas

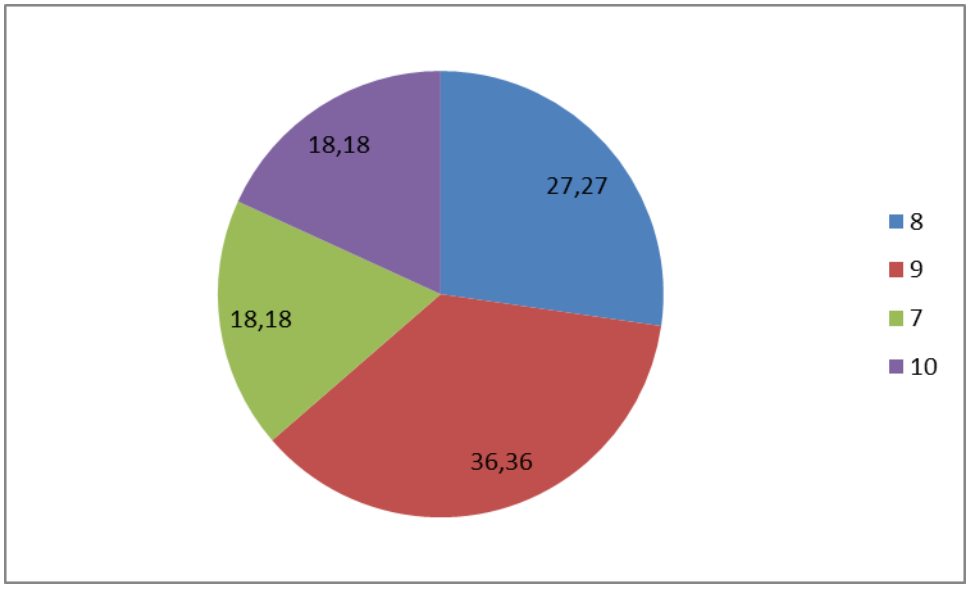

Figura 7. Pregunta 1 Compendio Buenas Prácticas, evaluación de legibilidad
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

En promedio la legibilidad del Compendio de Buenas Prácticas es de 8.55 sobre 10; lo cual supera las expectativas dada la brecha semántica existente entre quien escribió el documento (Argentino) y quienes lo leyeron (Ecuatorianos). Los resultados indican que el texto fue altamente comprendido por los encuestados y en ningún caso el puntaje recibido fue menor a 7.

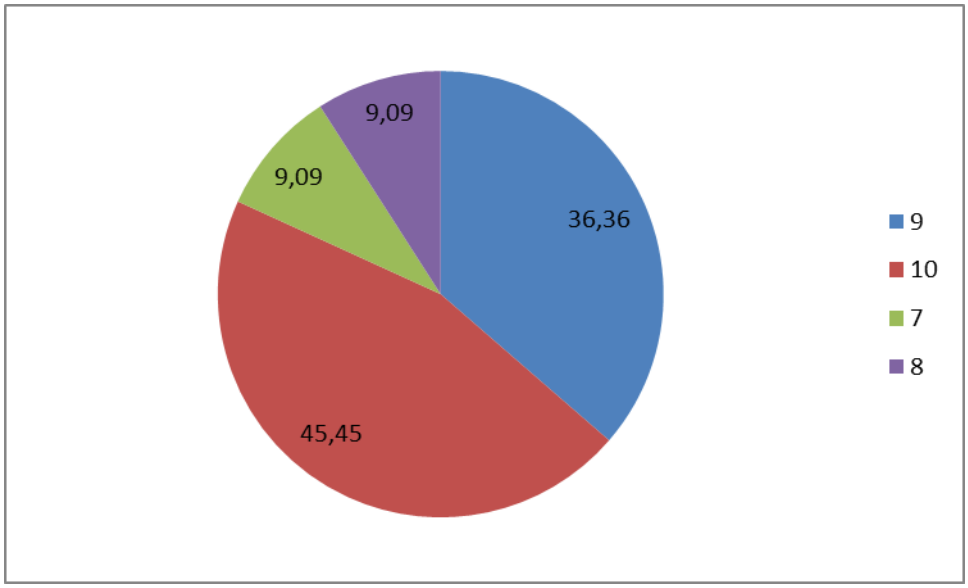

Figura 8. Pregunta 2 Compendio Buenas Prácticas, evaluación de aplicabilidad
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Según los encuestados, la aplicabilidad en promedio es de 9.09 sobre 10, lo cual indica que el Compendio de Buena Prácticas es considerado sumamente aplicable. En los resultados no se pueden observar ni uno solo caso en los que los encuestados indiquen que no es aplicable.

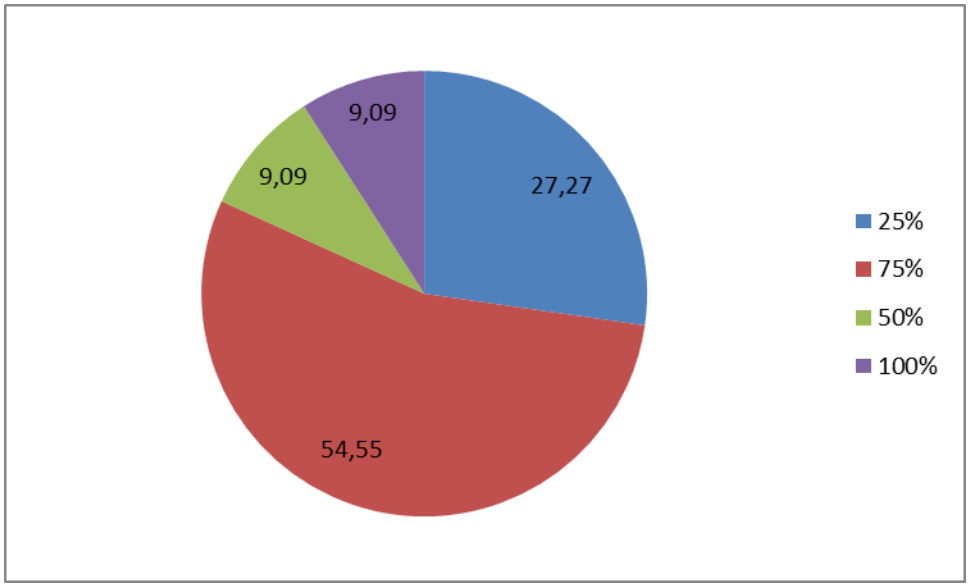

Figura 9. Pregunta 3 Compendio Buenas Prácticas, mejora de trabajo diario
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

En promedio los encuestados creen que aplicando el Compendio de Buenas Prácticas su trabajo diario mejorará en promedio en un 61% lo cual indica que el trabajo de los programadores mejorará bastante y que por lo tanto se desarrollarán aplicaciones más eficientes y que consumirán menos recursos.

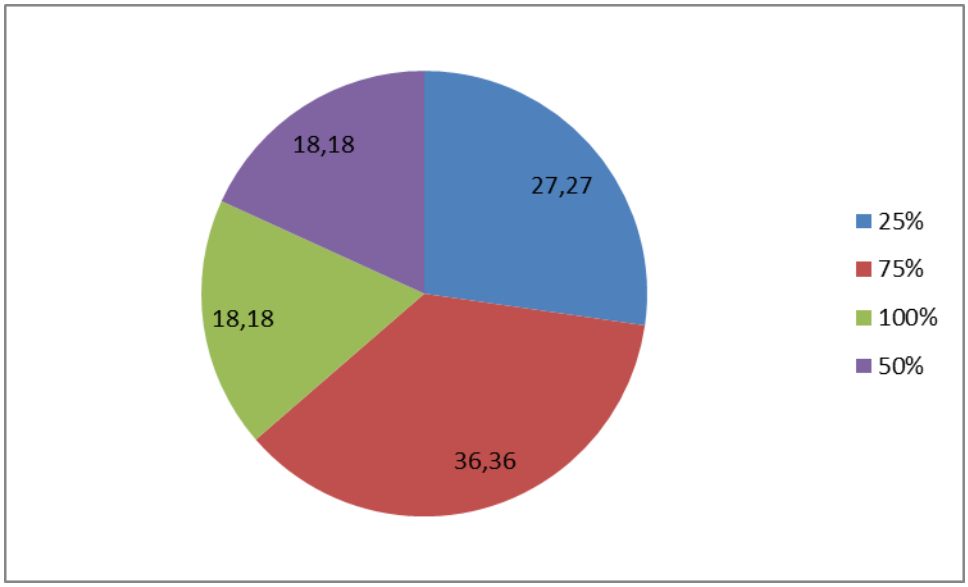

Figura 10. Pregunta 4 Compendio Buenas Prácticas, reducción de errores
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

En la misa proporción que los encuestados creen que mejorará su trabajo, también creen aplicando Buenas Prácticas se reducirán errores (61%), lo que por consiguiente permitirá se entreguen productos más eficientes y de mayor calidad. Ninguno de los encuestados indicó que su trabajo no tendrá mejoras al aplicar las Buenas Prácticas presentes en este documento.

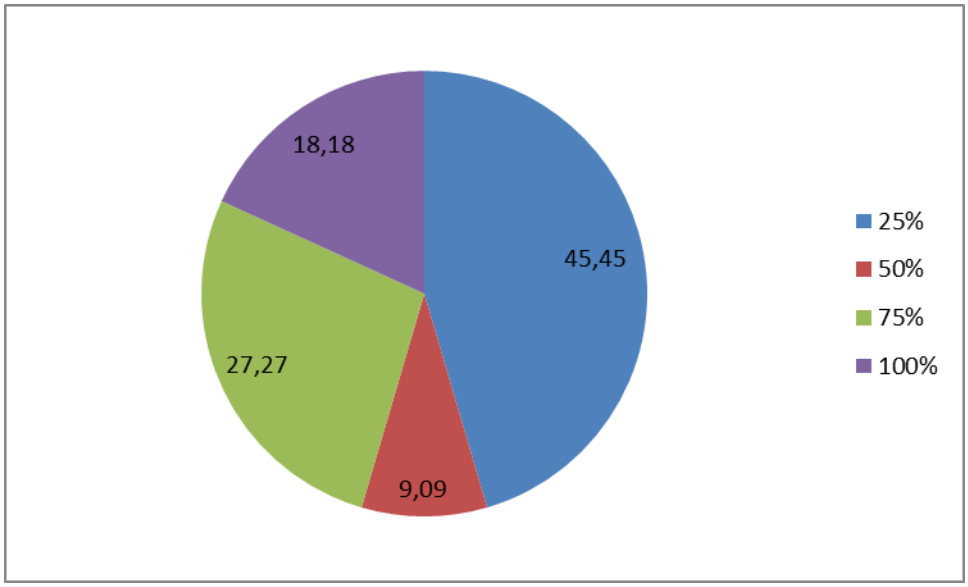

Figura 11. Pregunta 5 Compendio Buenas Prácticas, reducción de retrabajo
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Los encuestados indican que aplicando Buenas Prácticas en promedio el retrabajo se reducirá en un 55%, lo que permitirá reducir los tiempos de entrega de software y permitirá codificar aplicaciones con una mayor calidad desde un inicio dado que se insumirá menos tiempo al realizar esfuerzos adicionales necesarios para la corrección de alguna inconformidad en el código. Al reducirse los tiempos de entrega se podrá realizar una mejor planificación de los requerimientos de usuarios y optimizar el tiempo de los programadores, lo que por consiguiente permitirá atender mayor cantidad de requerimientos.

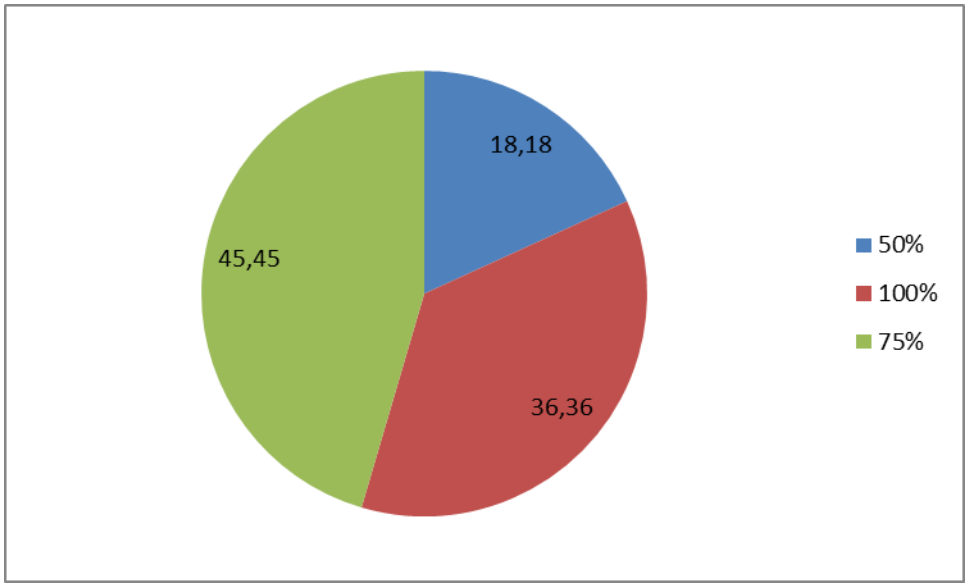

Figura 12. Pregunta 6 Compendio Buenas Prácticas, optimización de tiempos de ejecución
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Es muy importante indicar que todos los encuestados indican que se optimizarán los tiempos de ejecución. Según indican los encuestados, en promedio el tiempo de ejecución mejorará en un 80% lo cual brindará un uso más eficaz del motor de base de datos y permitirá ejecutar un mayor número de consultas de manera concurrente. Es decir, que los recursos del sistema trabajarán de tal manera que los resultados obtenidos se presentarán en menor tiempo y consumirán menos recursos del sistema y por ende se podrán procesar más peticiones al mismo tiempo de manera concurrente lo cual incrementará la cantidad total de peticiones que se podrán procesar.

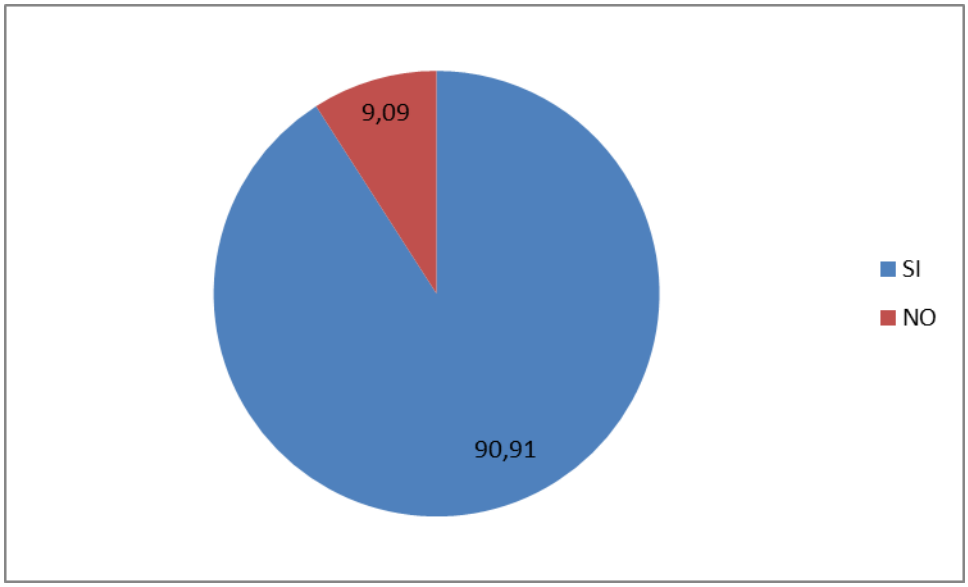

Figura 13. Pregunta 7 Compendio Buenas Prácticas, calidad de productos
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

El resultado obvio esperado era que el 100% de los encuestados hubieran indicado que aplicando Buenas Prácticas se obtienen productos de calidad, pero sin embargo en contraposición a lo esperado un porcentaje muy pequeño indicó que esto no es así. Los resultados obtenidos indican que en promedio el 91% de los encuestados afirma que al codificar utilizando Buenas Prácticas se obtienen programas de calidad.

6.1.1. Conclusiones para el Compendio de Buenas Prácticas

Los resultados obtenidos son alentadores porque indican que desde el punto de vista de los colegas encuestados; el compendio de Buenas Prácticas es muy bien comprendido y al ser aplicable permitirá realizar y construir consultas a las Bases de Datos que respondan en menos tiempo, consumiendo menos recursos y por consiguiente se utilizarán de manera

más eficaz los recursos del RDBMS permitiendo así se realicen mayor cantidad de transacciones de forma concurrente.

En cuanto al trabajo humano, este mejorará y por lo tanto será más eficiente porque se cometerán menos errores lo que permitirá reducir el tiempo en realizar esfuerzos adicionales para realizar correcciones reduciendo así el retrabajo. Todo esto permitirá a los programadores codificar programas con una mayor calidad final y por consiguiente ser más eficientes en su trabajo.

6.2. Resultados para Estándar de Programación

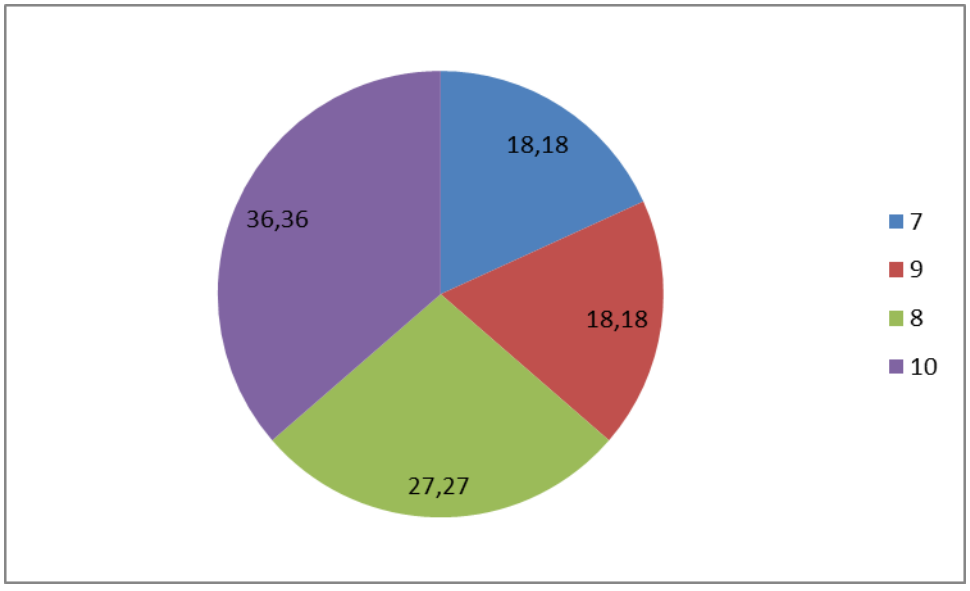

Figura 14. Pregunta 1 Estándar de Programación, evaluación de legibilidad
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

En promedio la legibilidad del Estándar de Programación es de 8.73 sobre 10; lo cual supera las expectativas dada la brecha semántica existente entre quien escribió el documento (Argentino) y quienes lo leyeron (Ecuatorianos). Los resultados indican que el texto fue altamente comprendido por los encuestados y en ningún caso el puntaje recibido fue menor a 7.

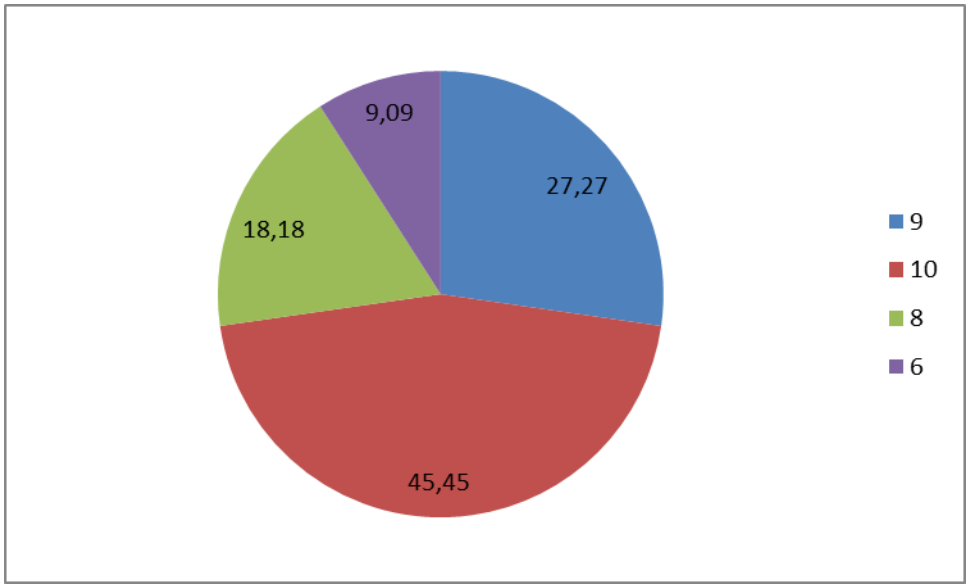

Figura 15. Pregunta 2 Estándar de Programación, evaluación de aplicabilidad
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Según los encuestados, la aplicabilidad en promedio es de 9 sobre 10, lo cual indica que el Estándar de Programación es considerado altamente aplicable. En los resultados no se pueden observar ni uno solo caso en los que los encuestados indiquen que no es aplicable.

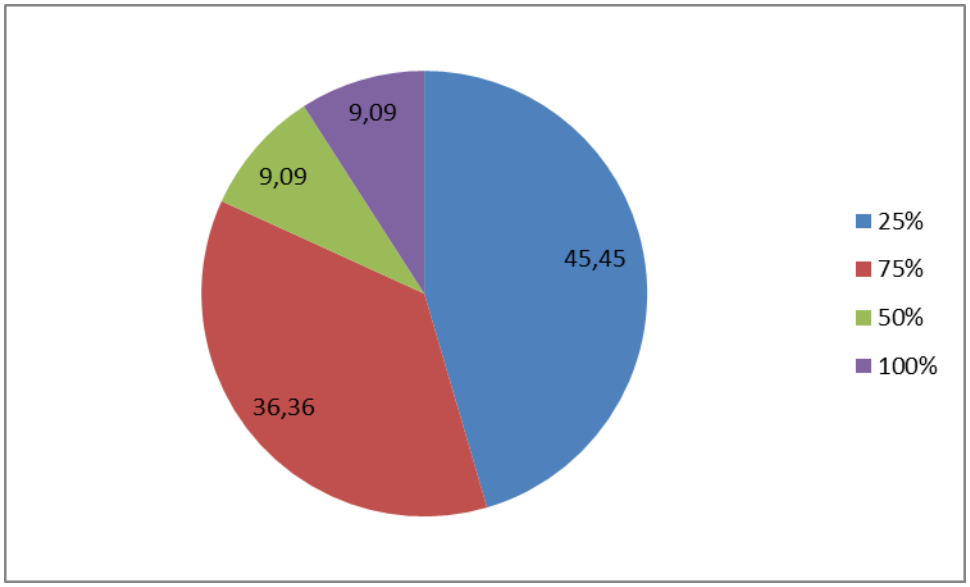

Figura 16. Pregunta 3 Estándar de Programación, mejora de trabajo diario
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

En promedio los encuestados creen que aplicando el Estándar de Programación su trabajo mejorará en un 52% lo cual indica que el trabajo de los programadores mejorará bastante y que por lo tanto se desarrollarán aplicaciones que cumplan con normas de codificación y permitirán que cualquier otro programador que a futuro necesite modificar dicho código pueda hacerlo de manera más rápida y sin incurrir en un gran esfuerzo para entender lo anteriormente desarrollado por otro programador.

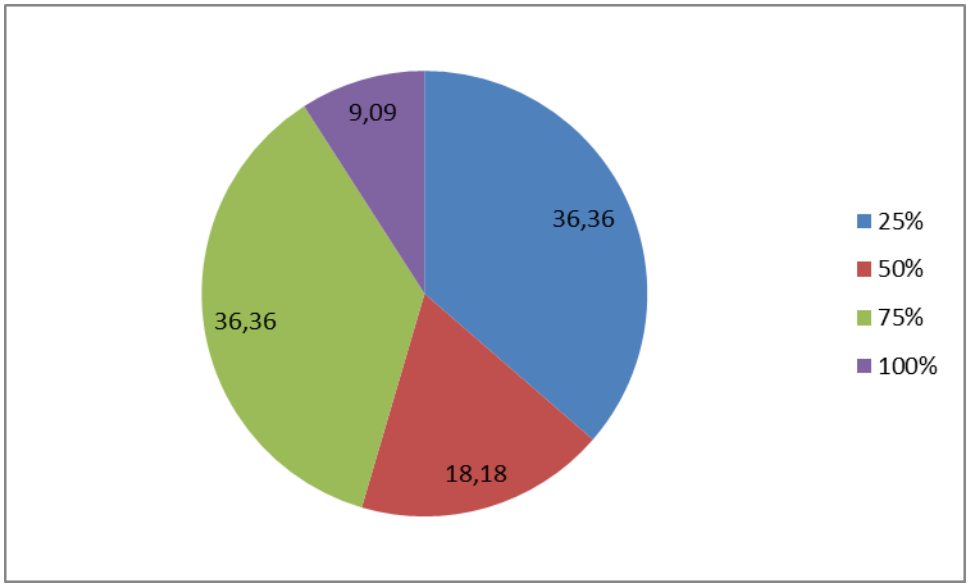

Figura 17. Pregunta 4 Estándar de Programación, reducción de errores
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Aplicando el Estándar de Programación según los encuestados en promedio indican se reducirán los errores en un 55%, lo que por consiguiente permitirá se entreguen productos que al cumplir con normas de codificación permitirán a la empresa poder aplicar a las distintas certificaciones de calidad que actualmente se exigen a nivel mundial. Ninguno de los encuestados indicó que su trabajo no tendrá mejoras al aplicar el Estándar de Programación presente en este documento.

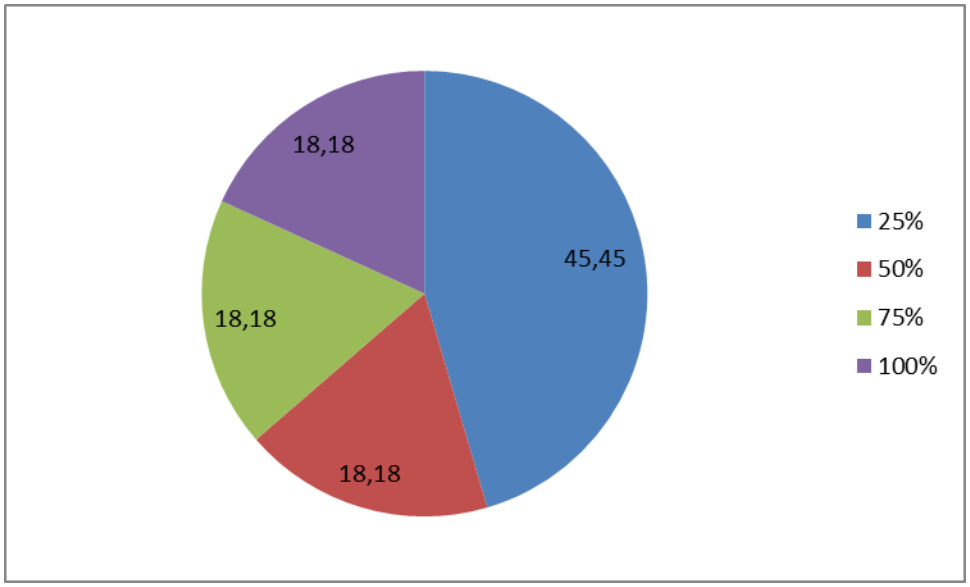

Figura 18. Pregunta 5 Estándar de Programación, reducción de retrabajo
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Los encuestados indican que aplicando el Estándar de Programación en promedio el retrabajo se reducirá en un 52%, lo que permitirá reducir los tiempos de entrega de software y permitirá codificar aplicaciones con una mayor calidad desde un inicio dado que se insumirá menos tiempo al realizar esfuerzos adicionales necesarios para la corrección de alguna inconformidad en el código. Al reducirse los tiempos de entrega se podrá realizar una mejor planificación de los requerimientos de usuarios y optimizar el tiempo de los programadores, lo que por consiguiente permitirá atender mayor cantidad de requerimientos.

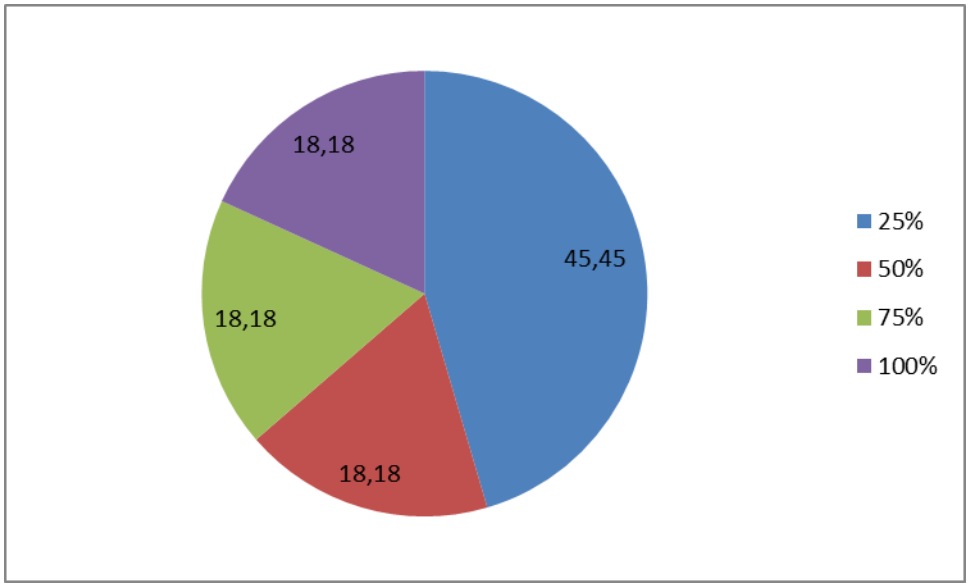

Figura 19. Pregunta 6 Estándar de Programación, optimización de tiempos de ejecución
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

Según los encuestados, al aplicar el Estándar de Programación en promedio los tiempos de ejecución mejorarán en un 52% lo cual optimizará el uso de la System Global Area del motor de Base de Datos y por lo tanto esto permitirá tener una mayor cantidad de sentencias iguales en memoria, lo que por consiguiente contribuirá a reducir los tiempos de ejecución ya que cada vez no se deberán realizar nuevamente los pasos de validación descriptos en el capítulo 4 y se encontrarán mayor cantidad de sentencias en la SGA.

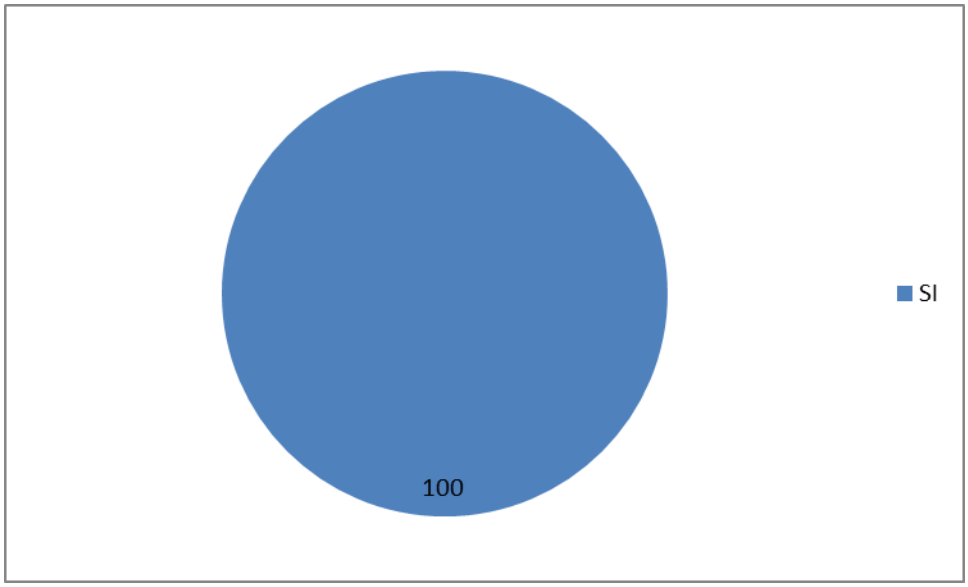

Figura 20. Pregunta 7 Estándar de Programación, calidad de productos
Fuente: Autoría personal en base a los resultados de la encuesta que se encuentra en Anexos

El resultado habla por sí solo y es contundente ya que todos los encuestados coinciden que aplicando el Estándar de Programación se obtendrán productos de mayor calidad y por consiguiente se obtendrá código más fácil lectura y uniformidad.

6.2.1. Conclusiones para el Estándar de Programación

La aplicación del Estándar de Programación garantiza uniformidad de código, permitiendo generar código que resulte fácil de seguir y que sea más entendible para todos los programadores. Esto permite a los programadores reducir el tiempo que emplean para entender el código desarrollado anteriormente por otros, lo que contribuye a disminuir los tiempos al tener que modificar dicho código y también en consecuencia se disminuye el retrabajo dado que se emplea menos tiempo al realizar esfuerzos adicionales necesarios

para la corrección de alguna inconformidad en el código o simplemente para realizar alguna actualización al mismo.

La alineación a un estándar permite el diseño, desarrollo y aplicación de un marco de trabajo, que entre otras cosas, contribuye a estructurar y ordenar tanto el código como las actividades de los programadores. Esto permitirá realizar una mejor planificación de los recursos tanto humanos como computacionales.

6.3. Recomendaciones

Tanto el Estándar de Programación como el Compendio de Buenas Prácticas son considerados legibles y aplicables por los colegas encuestados, no hubo ninguna persona encuestada que manifestara lo contrario, esto indica que es bien aceptado y por ende puede ponerse en práctica. Lamentablemente en las empresas que se propuso el mismo, no se tuvo la llegada, fuerza y/o los contactos suficientes como poder hacerlo. Por lo tanto, la mayor recomendación es ponerlos en práctica y así luego poder medir los resultados de su aplicación.

Para poder avalar los resultados obtenidos en las encuestas, se debe realizar una medición antes y otra después de su aplicación para así poder realizar las comparaciones y cruces necesarios que permitan afirmar lo expuesto en este documento.

La aplicación del Estándar de Programación permitirá a las empresas poder aplicar a las distintas certificaciones de calidad que actualmente se exigen a nivel mundial y es por eso que debe alentarse su uso. Por lo tanto se debe concientizar a las mismas de este beneficio.

Tanto el Compendio de Buenas Prácticas como el Estándar de Programación pueden aplicarse por separado, pero para obtener mejores resultados se recomienda que ambos sean aplicados al mismo tiempo. Puede implementarse un piloto con programadores experimentados y luego de verificados los resultados obtenidos, puede hacer extensiva su aplicación al resto de programadores. Para que los resultados obtenidos luego de su aplicación sean óptimos, se tiene que tener en cuenta la curva de aprendizaje que tendrán los programadores ya que deberán adaptarse a una nueva manera de programar y de pensar para escribir código que cumpla con las especificaciones y cuya ejecución resulte con una mayor performance.

La aplicación del Compendio de Buenas Prácticas y del Estándar de Programación no genera costo monetario extra para las empresas, el único costo es el tiempo que tomará a los programadores aprenderlos y adaptarse a los mismos.

La aplicación del Compendio de Buenas Prácticas y del Estándar de Programación puede hacerse en empresas de distintos tamaños (grandes, medianas y pequeñas) y no necesariamente en empresas con una amplia planta de programadores. Dada la curva de aprendizaje de los programadores, se debe verificar el impacto que tendrá su aplicación en

planteles de distintos tamaños. Posiblemente en planteles de menor tamaño la implementación sea más rápida o al menos pueda realizarse con mayor seguimiento.

Sería conveniente que en la aplicación tanto del Compendio de Buenas Prácticas como del Estándar de Programación de realizarse de forma escalonada se realice primero con aquellos programadores que tienen mayor experiencia y con un nivel de seniority de senior o al menos de semi senior y dejar por último a los programadores junior.

ANEXOS

Anexo 1: Encuesta sobre el Compendio Buenas Practicas para Oracle PL/SQL

Luego de haber leído el Compendio de Buenas Prácticas para Oracle PL/SQL que se encuentra explicado en el capítulo 4, por favor conteste las siguientes preguntas.

1. Del 1 al 10, siendo 1 malo y 10 excelente, evalúe la legibilidad

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

2. Del 1 al 10, siendo 1 malo y 10 excelente, evalúe la aplicabilidad

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

3. ¿Cuánto cree que mejorará su trabajo diario?

0%	**25%**	**50%**	**75%**	**100%**

4. ¿Cuánto cree se reducirán los errores?

0%	**25%**	**50%**	**75%**	**100%**

5. ¿Cuánto cree se reducirá el retrabajo?

0%	**25%**	**50%**	**75%**	**100%**

6. ¿Cuánto cree se optimizarán los tiempos de ejecución?

0%	**25%**	**50%**	**75%**	**100%**

7. ¿Con la aplicación de Buena Prácticas se logran productos de Calidad?

SI	**NO**	**NS/NC**

Anexo 2: Encuesta sobre el Estándar de Programación para Oracle PL/SQL

Luego de haber leído el Estándar de Programación para Oracle PL/SQL que se encuentra explicado en el capítulo 5, por favor conteste las siguientes preguntas.

1. Del 1 al 10, siendo 1 malo y 10 excelente, evalúe la legibilidad

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

2. Del 1 al 10, siendo 1 malo y 10 excelente, evalúe la aplicabilidad

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

3. ¿Cuánto cree que mejorará su trabajo diario?

0%	**25%**	**50%**	**75%**	**100%**

4. ¿Cuánto cree se reducirán los errores?

0%	**25%**	**50%**	**75%**	**100%**

5. ¿Cuánto cree se reducirá el retrabajo?

0%	**25%**	**50%**	**75%**	**100%**

6. ¿Cuánto cree se optimizarán los tiempos de ejecución?

0%	**25%**	**50%**	**75%**	**100%**

7. ¿Con la aplicación del Standard se logran productos de Calidad?

SI	**NO**	**NS/NC**

BIBLIOGRAFÍA

1. David M. Kroenke (2003), Procesamiento de bases de datos: fundamentos, diseño e implementación, Pearson Educación (2003).
2. Javier Morales Carreras (2013), Optimización SQL en Oracle: Una Guía Práctica, Detallada y Completa Sobre Cómo Implementar y Explotar Bases de Datos Oracle de Forma Eficiente, Createspace Independent Pub (2013).
3. Steven Feuerstein (2008), Oracle PL/SQL Best Practices, O'Reilly (2008).
4. ANSI/ISO/IEC 9075-2:2003, Information technology—Database languages—SQL—Part 2: Foundation (SQL/Foundation).
5. ANSI/ISO/IEC 9075-3:2003, Information technology—Database languages—SQL—Part 3: Call-Level Interface (SQL/CLI).
6. ANSI/ISO/IEC 9075-4:2003, Information technology—Database languages—SQL—Part 4: Persistent Stored Modules (SQL/PSM).
7. ANSI/ISO/IEC 9075-9:2003, Information technology—Database languages—SQL—Part 9: Management of External Data (SQL/MED).
8. ANSI/ISO/IEC 9075-10:2003, Information technology—Database languages—SQL—Part 10: Object Language Bindings (SQL/OLB).
9. ANSI/ISO/IEC 9075-11:2003, Information technology—Database languages—SQL—Part 11: Information and Definition Schemas (SQL/Schemata).

TRABAJOS CITADOS

1. Borland Software Corporation. (30 de Diciembre de 2013). *microfocus.com*. Obtenido de https://www.microfocus.com/about/press-room/article/2013/website-delays-cause-customer-and-revenue-drop-off-despite-strong-popularity-of-online-retail-promotions#
2. Bryan McQuade. (8 de Agosto de 2013). *webmasters.googleblog.com*. Obtenido de https://webmasters.googleblog.com/2013/08/making-smartphone-sites-load-fast.html
3. Globe Newsire. (19 de Julio de 2011). *globenewswire.com*. Obtenido de https://globenewswire.com/news-release/2011/07/19/451522/226802/en/New-Study-Reveals-the-Mobile-Web-Disappoints-Global-Consumers.html.
4. Joshua Bixby. (15 de Junio de 2010). *webperformancetoday.com*. Obtenido de http://www.webperformancetoday.com/2010/06/15/everything-you-wanted-to-know-about-web-performance/
5. Matt Watson. (29 de 05 de 2015). apmdigest.com. Obtenido de http://www.apmdigest.com/website-response-time-3
6. Robert Anderson. (29 de 05 de 2015). apmdigest.com. Obtenido de http://www.apmdigest.com/website-response-time-
7. Rothstein, Russell. (29 de 10 de 2016). itcentralstation.com. Obtenido de https://itcs-data.s3-us-west 2.amazonaws.com/pc_docs/1555/docs/Application_Performance_Management_%28APM%29_Report_from_IT_Central_Station_2016-10-29.pdf?AWSAccessKeyId=AKIAI3XZJNM4C37I3WLQ&Expires=1479311434&Signature=NOlJLdQ7MJhfaoeB5I6ng6i2ATA%3D
8. Web Performance Guru. (20 de Julio de 2011). *webperformanceguru.wordpress.com*. Obtenido de https://webperformanceguru.wordpress.com/2011/07/20/compuware-gomez-infographic-%E2%80%93-summary-of-key-mobile-survey-findings/
9. Work, Sean. (s.f.). *kissmetrics.com*. Obtenido de https://blog.kissmetrics.com/loading-time/